Microsoft Azure Well-Architected Framework

Article • 03/28/2023

The Azure Well-Architected Framework is a set of guiding tenets that you can use to improve the quality of a workload. The framework consists of five pillars of architectural excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

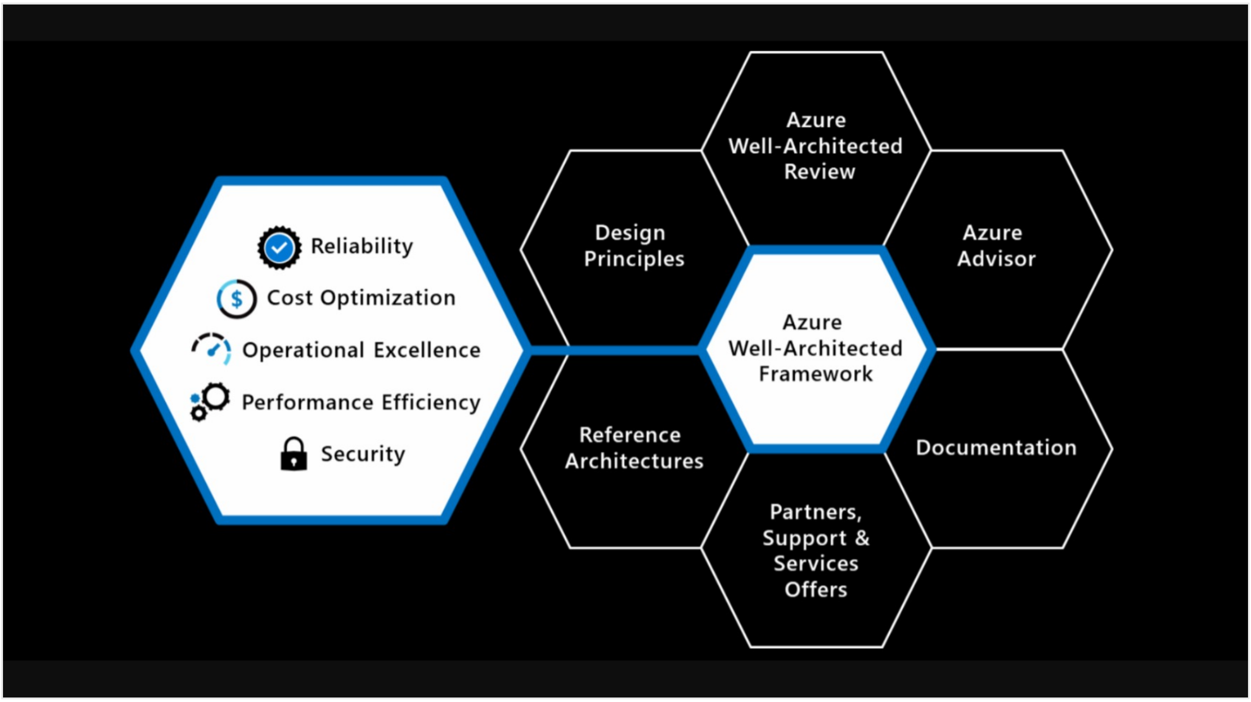
Incorporating these pillars helps produce a high quality, stable, and efficient cloud architecture:

Pillar	Description
Reliability	The ability of a system to recover from failures and continue to function.
Security	Protecting applications and data from threats.
Cost optimization	Managing costs to maximize the value delivered.
Operational excellence	Operations processes that keep a system running in production.
Performance efficiency	The ability of a system to adapt to changes in load.

To learn about how to architect successful workloads on Azure by using the Well-Architected Framework, watch this video: https://learn.microsoft.com/shows/azure-enablement/architect-successful-workloads-on-azure--introduction-ep-1-well-architected-series/player

Overview

The following diagram is a high-level overview of the Azure Well-Architected Framework:

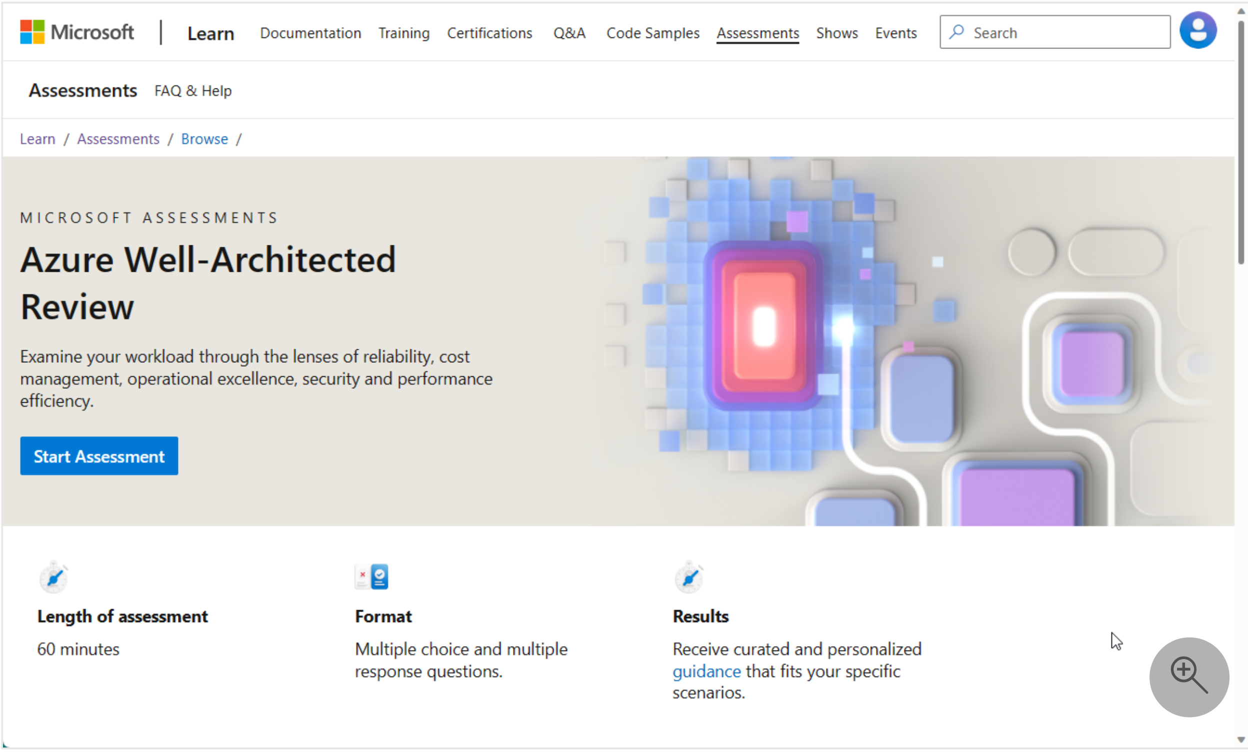


In the center is the *Well-Architected Framework*, which includes the five pillars of architectural excellence. Surrounding the Well-Architected Framework are six supporting elements:

- Azure Well-Architected Review
- Azure Advisor
- Documentation
- Partners, Support, and Services Offers
- Reference architectures
- Design principles

Assess your workload

To assess your workload using the tenets found in the Microsoft Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.



We also recommend that you use *Azure Advisor* and *Advisor Score* to identify and prioritize opportunities to improve the posture of your workloads. Both services are free to all Azure users and align to the five pillars of the Well-Architected Framework:

- *Azure Advisor* is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. It analyzes your resource configuration and usage telemetry. It recommends solutions that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your Azure resources. Learn more about Azure Advisor.

- *Advisor Score* is a core feature of Azure Advisor that aggregates Advisor recommendations into a simple, actionable score. This score enables you to tell at a glance if you're taking the necessary steps to build reliable, secure, and cost-efficient solutions. It helps to prioritize the actions that yield the biggest improvement to the posture of your workloads. The Advisor Score consists of an overall score, which can be further broken down into five category scores corresponding to each of the Well-Architected pillars. Learn more about Advisor Score.

Reliability

A reliable workload is both *resilient* and *available*. Resiliency is the ability of the system to recover from failures and continue to function. The goal of resiliency is to return the application to a fully functioning state after a failure occurs. Availability is whether your users can access your workload when they need to.

For more information about resiliency, watch the following video that shows you how to start improving the reliability of your Azure workloads: https://learn.microsoft.com/shows/azure-enablement/start-improving-the-reliability-of-your-azure-workloads--reliability-ep-1--well-architected-series/player

Reliability guidance

The following resources offer guidance on designing and improving reliable Azure applications:

- Reliability design principles
- Design patterns for resiliency
- Best practices:
  - Transient fault handling
  - Retry guidance for specific services

For an overview of reliability principles, see Reliability design principles.

Security

Think about security throughout the entire lifecycle of an application, from design and implementation to deployment and operations. The Azure platform provides protections against various threats, such as network intrusion and DDoS attacks. You still need to build security into your application and into your DevOps processes.

Learn to ask the right questions about secure application development on Azure by watching the following video: https://learn.microsoft.com/shows/azure-enablement/ask-the-right-questions-about-secure-application-development-on-azure/player

Security guidance

Consider the following broad security areas:

- Identity management
- Protect your infrastructure
- Application security
- Data sovereignty and encryption
- Security resources

For more information, see Overview of the security pillar.

Cost optimization

When you design a cloud solution, focus on generating incremental value early. Apply the principles of *Build-Measure-Learn* to accelerate your time to market while avoiding capital-intensive solutions. See What is the build-measure-learn feedback loop.

For more information, see Cost optimization and watch the following video on how to start optimizing your Azure costs: https://learn.microsoft.com/shows/azure-enablement/start-optimizing-your-azure-costs--cost-optimization-ep-1--well-architected-series/player

Cost guidance

The following resources offer cost optimization guidance as you develop the Well-Architected Framework for your workload:

- Review cost principles
- Develop a cost model
- Create budgets and alerts
- Review the cost optimization checklist

For a high-level overview, see Overview of the cost optimization pillar.

Operational excellence

Operational excellence covers the operations and processes that keep an application running in production. Deployments must be reliable and predictable. Automate deployments to reduce the chance of human error. Fast and routine deployment processes don't slow down the release of new features or bug fixes. Equally important, you must be able to quickly roll back or roll forward if an update has problems.

For more information, watch the following video about bringing security into your DevOps practice on Azure: https://learn.microsoft.com/shows/azure-enablement/devsecops-bringing-security-into-your-devops-practice-on-azure/player

Operational excellence guidance

The following resources provide guidance on designing and implementing DevOps practices for your Azure workload:

- Operational excellence patterns

- Best practices: Monitoring and diagnostics guidance

For a high-level summary, see Overview of the operational excellence pillar.

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. The main ways to achieve performance efficiency include using scaling appropriately and implementing PaaS offerings that have scaling built in.

For more information, watch Performance Efficiency: Fast & Furious: Optimizing for Quick and Reliable VM Deployments.

Performance efficiency guidance

The following resources offer guidance on how to design and improve the performance efficiency posture of your Azure workload:

- Performance efficiency patterns
- Best practices:
  - Autoscaling
  - Background jobs
  - Caching
  - CDN
  - Data partitioning

For a high-level overview, see Overview of the performance efficiency pillar.

Next steps

Learn more about:

- Azure Well-Architected Review
- Well-Architected Series
- Introduction to the Microsoft Azure Well-Architected Framework
- Microsoft Defender for Cloud
- Cloud Adoption Framework
- Azure Deployment Environments
- Microsoft Dev Box

Microsoft Azure Well-Architected Framework

Article • 03/28/2023

The Azure Well-Architected Framework is a set of guiding tenets that you can use to improve the quality of a workload. The framework consists of five pillars of architectural excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

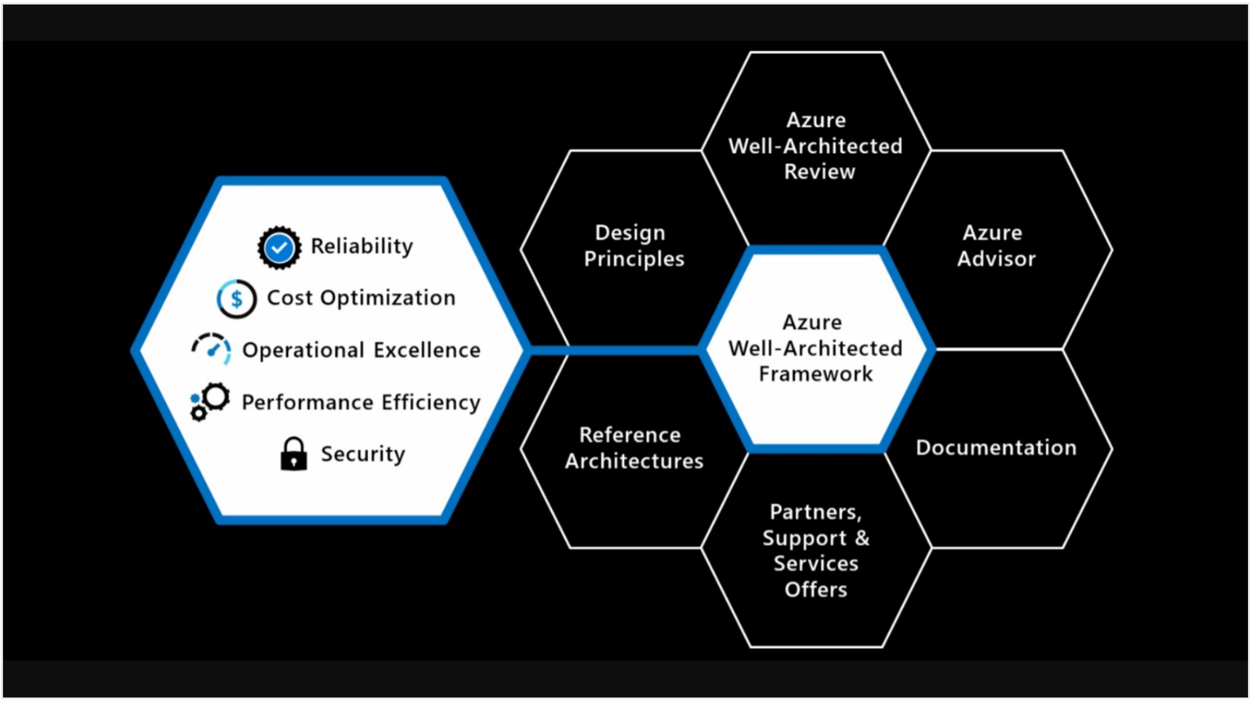
Incorporating these pillars helps produce a high quality, stable, and efficient cloud architecture:

Pillar	Description
Reliability	The ability of a system to recover from failures and continue to function.
Security	Protecting applications and data from threats.
Cost optimization	Managing costs to maximize the value delivered.
Operational excellence	Operations processes that keep a system running in production.
Performance efficiency	The ability of a system to adapt to changes in load.

To learn about how to architect successful workloads on Azure by using the Well-Architected Framework, watch this video: https://learn.microsoft.com/shows/azure-enablement/architect-successful-workloads-on-azure--introduction-ep-1-well-architected-series/player

Overview

The following diagram is a high-level overview of the Azure Well-Architected Framework:

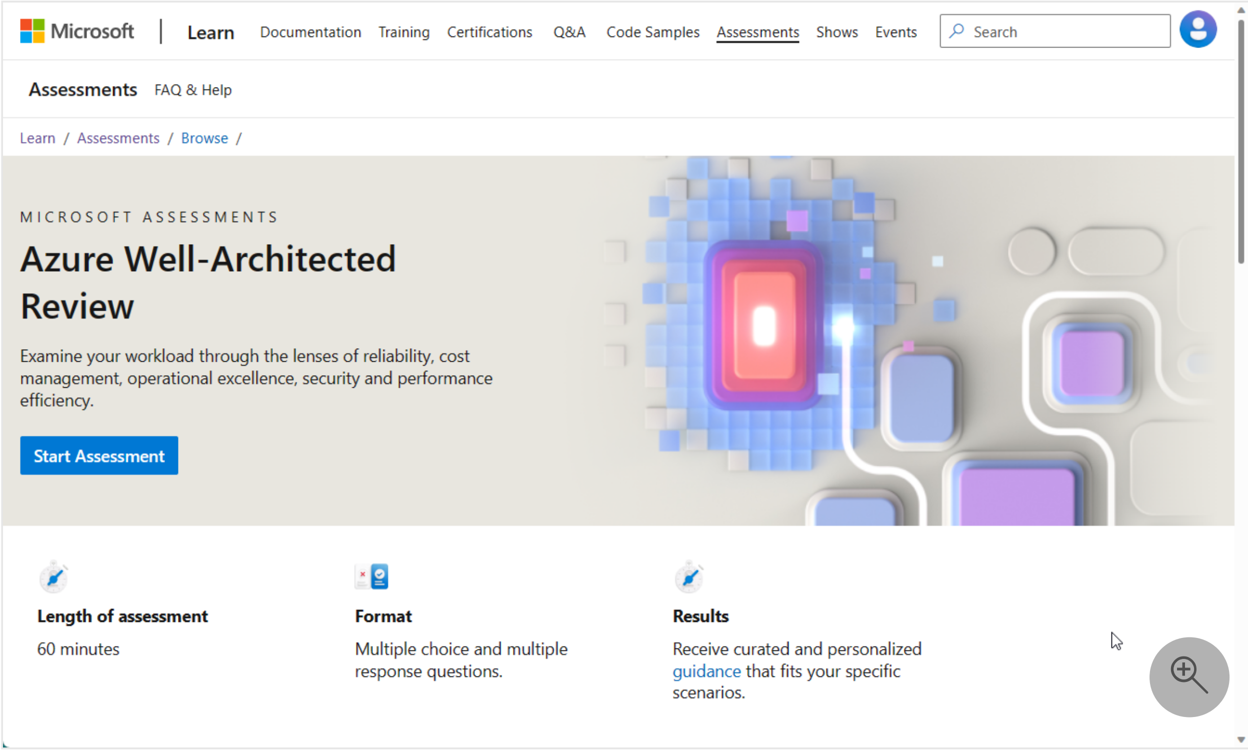


In the center is the *Well-Architected Framework*, which includes the five pillars of architectural excellence. Surrounding the Well-Architected Framework are six supporting elements:

- Azure Well-Architected Review
- Azure Advisor
- Documentation
- Partners, Support, and Services Offers
- Reference architectures
- Design principles

Assess your workload

To assess your workload using the tenets found in the Microsoft Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.



We also recommend that you use *Azure Advisor* and *Advisor Score* to identify and prioritize opportunities to improve the posture of your workloads. Both services are free to all Azure users and align to the five pillars of the Well-Architected Framework:

- *Azure Advisor* is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. It analyzes your resource configuration and usage telemetry. It recommends solutions that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your Azure resources. Learn more about Azure Advisor.

- *Advisor Score* is a core feature of Azure Advisor that aggregates Advisor recommendations into a simple, actionable score. This score enables you to tell at a glance if you're taking the necessary steps to build reliable, secure, and cost-efficient solutions. It helps to prioritize the actions that yield the biggest improvement to the posture of your workloads. The Advisor Score consists of an overall score, which can be further broken down into five category scores corresponding to each of the Well-Architected pillars. Learn more about Advisor Score.

Reliability

A reliable workload is both *resilient* and *available*. Resiliency is the ability of the system to recover from failures and continue to function. The goal of resiliency is to return the application to a fully functioning state after a failure occurs. Availability is whether your users can access your workload when they need to.

For more information about resiliency, watch the following video that shows you how to start improving the reliability of your Azure workloads: https://learn.microsoft.com/shows/azure-enablement/start-improving-the-reliability-of-your-azure-workloads--reliability-ep-1--well-architected-series/player

Reliability guidance

The following resources offer guidance on designing and improving reliable Azure applications:

- Reliability design principles
- Design patterns for resiliency
- Best practices:
  - Transient fault handling
  - Retry guidance for specific services

For an overview of reliability principles, see Reliability design principles.

Security

Think about security throughout the entire lifecycle of an application, from design and implementation to deployment and operations. The Azure platform provides protections against various threats, such as network intrusion and DDoS attacks. You still need to build security into your application and into your DevOps processes.

Learn to ask the right questions about secure application development on Azure by watching the following video: https://learn.microsoft.com/shows/azure-enablement/ask-the-right-questions-about-secure-application-development-on-azure/player

Security guidance

Consider the following broad security areas:

- Identity management
- Protect your infrastructure
- Application security
- Data sovereignty and encryption
- Security resources

For more information, see Overview of the security pillar.

Cost optimization

When you design a cloud solution, focus on generating incremental value early. Apply the principles of *Build-Measure-Learn* to accelerate your time to market while avoiding capital-intensive solutions. See What is the build-measure-learn feedback loop.

For more information, see Cost optimization and watch the following video on how to start optimizing your Azure costs: https://learn.microsoft.com/shows/azure-enablement/start-optimizing-your-azure-costs--cost-optimization-ep-1--well-architected-series/player

Cost guidance

The following resources offer cost optimization guidance as you develop the Well-Architected Framework for your workload:

- Review cost principles
- Develop a cost model
- Create budgets and alerts
- Review the cost optimization checklist

For a high-level overview, see Overview of the cost optimization pillar.

Operational excellence

Operational excellence covers the operations and processes that keep an application running in production. Deployments must be reliable and predictable. Automate deployments to reduce the chance of human error. Fast and routine deployment processes don't slow down the release of new features or bug fixes. Equally important, you must be able to quickly roll back or roll forward if an update has problems.

For more information, watch the following video about bringing security into your DevOps practice on Azure: https://learn.microsoft.com/shows/azure-enablement/devsecops-bringing-security-into-your-devops-practice-on-azure/player

Operational excellence guidance

The following resources provide guidance on designing and implementing DevOps practices for your Azure workload:

- Operational excellence patterns

- Best practices: Monitoring and diagnostics guidance

For a high-level summary, see Overview of the operational excellence pillar.

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. The main ways to achieve performance efficiency include using scaling appropriately and implementing PaaS offerings that have scaling built in.

For more information, watch Performance Efficiency: Fast & Furious: Optimizing for Quick and Reliable VM Deployments.

Performance efficiency guidance

The following resources offer guidance on how to design and improve the performance efficiency posture of your Azure workload:

- Performance efficiency patterns
- Best practices:
  - Autoscaling
  - Background jobs
  - Caching
  - CDN
  - Data partitioning

For a high-level overview, see Overview of the performance efficiency pillar.

Next steps

Learn more about:

- Azure Well-Architected Review
- Well-Architected Series
- Introduction to the Microsoft Azure Well-Architected Framework
- Microsoft Defender for Cloud
- Cloud Adoption Framework
- Azure Deployment Environments
- Microsoft Dev Box

Reliability documentation

Apply resiliency principles to your architecture to ensure an application returns to a fully functioning state after a failure occurs. A reliable workload is both resilient and available.

Key points

QUICKSTART

Principles

Design checklist

Testing checklist

Monitoring checklist

Reliability patterns

Take the Azure Well-Architected Review

TRAINING

Reliability

Resilience in Azure whitepaper

VIDEO

Inside Azure Datacenter Architecture with Mark Russinovich

Region and availability zone fundamentals

CONCEPT

Explore business continuity management in Azure

Achieve reliability through regions and availability zones

Understand Azure service offerings across region types

Consider Azure services that support availability zones

Understand region and availability zone terminology

Design for availability and resiliency

GET STARTED

Consider target and non-functional requirements

Use availability zones within a region

Respond to failure

Build resiliency with failure mode analysis

Understand the impact of dependencies

CONCEPT

Understand design best practices

Ensure connectivity

Use zone-aware services

Design for scalability

TRAINING

Design a geographically distributed application

VIDEO

Zone to zone disaster recovery with Azure Site Recovery

Test for availability and resiliency

OVERVIEW

Resiliency testing

Backup and disaster recovery

Error handling

Chaos engineering

CONCEPT

Understand testing best practices

Verify workload performance under failure

Verify workload performance under failure

Design a backup strategy

Design a disaster recovery strategy

VIDEO

Using availability zones for Kubernetes cluster

Monitor reliability

OVERVIEW

Diagnose application health

Use health models for reliability

CONCEPT

Understand monitoring best practices

Implement health probes and check functions

Check long-running workflows

Maintain application logs

VIDEO

SQL high availability by service tier

Reliability tools and services

REFERENCE

Azure Advisor

Advisor Score

Azure Kubernetes Service (AKS)

Azure SQL Database

Azure API Management

Application Gateway (V2)

Key Vault

Key Vault

Azure Backup

Azure Site Recovery

Reliability APIs

REFERENCE

API Management

Recovery Services vault

Backup REST API

Site Recovery REST API

Overview of the reliability pillar

Article • 05/30/2023

Reliability ensures that your application can meet the commitments you make to your customers. Architecting reliability into your application framework ensures that your workloads are available and can recover from failures at any scale.

Building for reliability includes:

- Ensuring a highly available architecture
- Recovering from failures such as data loss, major downtime, or ransomware incidents

To assess the reliability of your workload using the tenets found in the Microsoft Azure Well-Architected Framework, take the Microsoft Azure Well-Architected Review.

For more information, explore the following video on diving deeper into Azure workload reliability:

https://learn.microsoft.com/_themes/docs.theme/master/en-us/_themes/global/video-embed.html?show=azure-enablement&ep=diving-deeper-into-azure-workload-reliability-part-2--reliability-ep-2--well-architected&locale=en-us&embedUrl=%2Fazure%2Fwell-architected%2Fresiliency%2Foverview

In traditional application development, there has been a focus on increasing the mean time between failures (MTBF). Effort was spent trying to prevent the system from failing. In cloud computing, a different mindset is required because of several factors:

- Distributed systems are complex. A failure at one point can potentially cascade throughout the system.
- Costs for cloud environments are kept low through commodity hardware. Occasional hardware failures must be expected.
- Applications often depend on external services. Those services might be temporarily unavailable or throttle high-volume users.
- Today's users expect an application to be available 24/7 without ever going offline.

All of these factors mean that cloud applications must be designed to expect occasional failures and recover from them. Azure has many resiliency features already built into the platform, such as these examples:

- Azure Storage, Azure SQL Database, and Azure Cosmos DB provide built-in data replication across availability zones and regions.

- Azure managed disks are automatically placed in different storage scale units to limit the effects of hardware failures.
- Virtual machines in an availability set are spread across several fault domains. A *fault domain* is a group of virtual machines that share a common power source and network switch. Spreading virtual machines across fault domains limits the effect of physical hardware failures, network outages, or power interruptions.
- *Availability zones* are physically separate locations within an Azure region. Each zone is composed of one or more datacenters equipped with independent power, cooling, and networking infrastructure. With availability zones, you can design and operate applications and databases that automatically transition between zones without interruption. This approach ensures reliability if one zone is affected. For more information, reference Azure regions and availability zones.

Even with these features, you still need to build resiliency into your application. Resiliency strategies can be applied at all levels of the architecture. Some mitigations are more tactical in nature, for example, retrying a remote call after a transient network failure. Other mitigations are more strategic, such as failing over the entire application to a secondary region.

Tactical mitigations can make a large difference. While it's rare for an entire region to experience a disruption, transient problems such as network congestion are more common. Target these issues first. Having the right monitoring and diagnostics is also important, both to detect failures when they happen, and to find the root causes.

When designing an application to be resilient, you must understand your availability requirements. How much downtime is acceptable? The amount of downtime is partly a function of cost. How much does potential downtime cost your business? How much should you invest in making the application highly available?

Topics and best practices

The reliability pillar covers the following topics and best practices to help you build a resilient workload:

Reliability article	Description
Reliability design principles	These critical principles are used as lenses to assess the reliability of an application deployed on Azure.
Design for reliability	Consider how systems use availability zones, perform scalability, respond to failure, and other strategies that optimize reliability in application design.

Reliability article	Description
Resiliency checklist for specific Azure services	Every technology has its own particular failure modes, which you must consider when designing and implementing your application. Use this checklist to review the resiliency considerations for specific Azure services.
Target functional and nonfunctional requirements	Target functional and nonfunctional requirements, such as availability targets and recovery targets, allow you to measure the uptime and downtime of your workloads. Having clearly defined targets is crucial to have a goal to work and measure against.
Resiliency and dependencies	Building failure recovery into the system should be part of the architecture and design phases from the beginning to avoid the risk of failure. Dependencies are required for the application to fully operate.
High availability using availability zones	Availability zones can be used to spread a solution across multiple zones within a region, allowing for an application to continue functioning when one zone fails.
Available services	Availability of services across Azure regions depends on a region's type. Azure's general policy on deploying services into any given region is primarily driven by region type, service categories, and customer demand.
Availability zone terminology	To better understand regions and availability zones in Azure, it helps to understand key terms or concepts.
Best practices for reliability in applications	During the architectural phase, focus on implementing practices that meet your business requirements, identify failure points, and minimize the scope of failures.
Testing for reliability	Regular testing should be performed as part of each major change to validate existing thresholds, targets, and assumptions.
Monitoring for reliability	Get an overall picture of application health. If something fails, you need to know *that* it failed, *when* it failed, and *why*.
Reliability patterns	Applications must be designed and implemented to maximize availability.

Next steps

Reliability design principles

Reliability design principles

Article • 11/30/2022

Building a reliable application in the cloud is different from traditional application development. Historically, you may have purchased levels of redundant higher-end hardware to minimize the chance of an entire application platform failing.

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component.

To assess your workload using the tenets found in the Azure Well-Architected Framework, reference the Microsoft Azure Well-Architected Review.

The following design principles provide:

- Context for questions
- Why a certain aspect is important
- How an aspect is applicable to Reliability

These critical design principles are used as lenses to assess the Reliability of an application deployed on Azure. These lenses provide a framework for the application assessment questions.

Design for business requirements

Reliability is a subjective concept. For an application to be appropriately reliable, it must reflect the business requirements surrounding it.

For example, a mission-critical application with a `99.999%` service level agreement (SLA) requires a higher level of reliability than another application with an SLA of `95%`.

Cost implications are inevitable when introducing greater reliability and high availability. This trade-off should be carefully considered.

Design for failure

Failure is impossible to avoid in a highly distributed and multi-tenant environment like Azure.

By anticipating failures, from individual components to entire Azure regions, you can develop a solution in a resilient way to increase reliability.

Observe application health

Before mitigating issues that impact application reliability, you must first detect these issues.

By monitoring the operation of an application relative to a healthy state, you can detect and predict reliability issues.

Monitoring allows you to take swift and remedial action.

Drive automation

One of the leading causes of application downtime is human error due to the deployment of insufficiently tested software or through misconfiguration.

To minimize the possibility and consequence of human errors, it's vital to strive for automation in all aspects of a cloud solution.

Automation improves:

- Reliability
- Automated testing
- Deployment
- Management

Design for self-healing

Self-healing describes the ability of a system to deal with failures automatically. Handling failures happens through pre-defined remediation protocols. These protocols connect to failure modes within the solution.

It's an advanced concept that requires a high level of system maturity with monitoring and automation.

From inception, self-healing should be an aspiration to maximize reliability.

Design for scale-out

Scale-out is a concept that focuses on the ability of a system to respond to demand through horizontal growth. As traffic grows, *more* resource units are added in parallel, instead of increasing the size of the existing resources.

Through scale units, a system can handle expected and unexpected traffic increases, essential to overall reliability.

Scale units further reduce the effects of a single resource failure.

Next step

Design

Design for reliability

Article • 05/30/2023

Reliable applications should maintain a predefined percentage of uptime, or *availability*. They should also balance between high resiliency, low latency, and cost, which is *high availability*. As important, applications should be able to recover from failures, which is *resiliency*.

Checklist

How have you designed your applications with reliability in mind?

- ✓ Define availability and recovery targets to meet business requirements.
- ✓ Build reliability and availability into your apps by gathering requirements.
- ✓ Ensure that application and data platforms meet your reliability requirements.
- ✓ Configure connection paths to promote availability.
- ✓ Use Azure availability zones where applicable to improve reliability and optimize costs.
- ✓ Ensure that your application architecture is resilient to failures.
- ✓ Know what happens if the requirements of service-level agreements (SLAs) are not met.
- ✓ Identify possible failure points in the system to build reliability.
- ✓ Ensure that applications can operate in the absence of their dependencies.

Azure services

- Azure Front Door
- Azure Traffic Manager
- Azure Load Balancer
- Azure NAT Gateway
- Azure Service Fabric
- Azure Kubernetes Service (AKS)
- Azure Site Recovery

Reference architecture

- Deploy highly available network virtual appliances
- Failure mode analysis for Azure applications
- Minimize coordination

Related Links

- Use platform as a service (PaaS) options
- Design to scale out
- Workload availability targets
- Building solutions for high availability using availability zones
- Make all things redundant

Next steps

Target functional and nonfunctional requirements

Target functional and nonfunctional requirements

Article • 05/30/2023

Target functional and nonfunctional requirements include *availability targets* and *recovery targets*. These requirements allow you to measure the uptime and downtime of your workloads. Having clearly defined targets is crucial to have a goal to work and measure against. There are many other requirements that you should consider that improve reliability requirements and meet business expectations.

Resiliency means the ability to recover from failures. *Availability* means to run in a healthy state without significant downtime. Building resiliency and availability into your apps begins with gathering requirements. For example, how much downtime is acceptable and how much does potential downtime cost your business?

Key points

- Determine the acceptable level of uptime for your workloads.
- Determine how long workloads can be unavailable and how much data is acceptable to lose during a disaster.
- Consider application and data platform requirements to improve resiliency and availability.
- Ensure connection availability and improve reliability with Azure services.
- Assess overall application health of workloads.

Availability targets

A service-level agreement (SLA) is an availability target that represents a commitment around performance and availability of the application. Understanding the SLA of individual components within the system is essential to define reliability targets. Knowing the SLA of dependencies also provides a justification for extra spending when making the dependencies highly available and with proper support contracts. Availability targets for any dependencies used by the application should be understood and ideally align with application targets should also be considered.

Understanding your availability expectations is vital to reviewing overall operations for the application. If you want to achieve an application service-level objective (SLO) of 99.999%, the level of inherent operational action required by the application is going to be far greater than if an SLO of 99.9% is the goal.

Monitoring and measuring application availability is vital to qualifying overall application health and progress towards defined targets. Make sure you measure and monitor key targets such as:

- Mean Time Between Failures (MTBF). The average time between failures of a particular component.
- Mean Time To Recover (MTTR). The average time it takes to restore a component after a failure.

Considerations for availability targets

Keep the following questions in mind.

Are SLAs/SLOs/SLIs for all applied dependencies understood?

Availability targets for any dependencies used by the application should be understood and ideally align with application targets. Make sure SLAs/SLOs/service level indicators (SLIs) for all applied dependencies are understood.

Has a composite SLA been calculated for the application and key scenarios using Azure SLAs?

A composite SLA captures the end-to-end SLA across all application components and dependencies. It's calculated using the individual SLAs of Azure services hosting application components. A composite SLA provides an important indicator of designed availability in relation to customer expectations and targets. Make sure that the composite SLA of all components and dependencies on the critical paths are understood. For more information, see Composite SLAs.

> ⓘ **Note**
>
> If you have contractual commitments to an SLA for your Azure solution, allowances on top of the Azure composite SLA must be made to accommodate outages caused by code-level issues and deployments. This fact is often overlooked. You might directly put the composite SLA forward to your customers.

Are availability targets considered while the system runs in disaster recovery mode?

Availability targets might or might not be applied when the system runs in disaster recovery mode. It depends on the application. If targets must also apply in a failure state, an *N+1 model* should be used to achieve greater availability and resiliency. In this scenario, N is the capacity needed to deliver required availability. There's also a cost implication. More resilient infrastructure is usually more expensive.

What are the consequences if availability targets aren't satisfied?

Are there any penalties, such as financial charges, associated with failing to meet SLA commitments? Other measures can be used to prevent penalties, but that also brings extra cost to operate the infrastructure. This fact has to be factored in and evaluated. The consequences if availability targets aren't satisfied should be fully understood. This approach also informs when to initiate a failover case.

Recovery targets

Recovery targets identify how long the workload can be unavailable and how much data is acceptable to lose during a disaster. Define target reports for the application and key scenarios. Target reports needed are:

- Recovery Time Objective (RTO). The maximum acceptable time an application is unavailable after an incident.
- Recovery Point Objective (RPO). The maximum duration of data loss that is acceptable during a disaster.

Recovery targets are nonfunctional requirements of a system and should be dictated by business requirements. Recovery targets should be defined in accordance to the required RTO and RPO targets for the workloads.

Meet application platform requirements

Azure application platform services offer resiliency features to support application reliability. They might only apply for a certain SKU and configuration or deployment. For example, they might apply if an SLA is dependent on the number of instances deployed or a certain feature enabled.

We recommend that you review the SLA for services used. For example, the Service Bus Premium SKU provides predictable latency and throughput to mitigate noisy neighbor scenarios. It also lets you automatically scale and replicate metadata to another Service Bus instance for failover purposes.

For more information, see Service Bus Premium and Standard messaging tiers.

Multiple and paired regions

An application platform should be deployed across multiple regions if the requirements dictate. Covering the requirements using zones is cheaper and less complex. Regional isolation should be an extra measure if the SLAs given by the single region cross-zone setup are insufficient or if required by a geographical spread of users.

The ability to respond to disaster scenarios for overall compute platform availability and application reliability depends on the use of multiple regions or other deployment locations.

Use paired regions that exist within the same geography and provide native replication features for recovery purposes, such as geo-redundant storage (GRS) asynchronous replication. If you plan maintenance, updates to a region are performed sequentially only. For more information, see Cross-region replication in Azure.

Availability zones and availability sets

Platform services that can use availability zones are deployed in either a zonal configuration in a particular zone, or in a zone-redundant configuration across multiple zones. For more information, see Building solutions for high availability using availability zones.

An availability set is a logical construct to inform Azure to distribute contained virtual machine instances across multiple fault and update domains within an Azure region. Availability zones elevate the fault level for virtual machines to a physical datacenter by allowing replica instances to be deployed across multiple datacenters within an Azure region.

Availability zones provide greater reliability than availability sets. There are performance and cost considerations where applications are extremely *chatty* across zones given the implied physical separation and inter-zone bandwidth charges. Azure Virtual Machines and Azure platform as a service (PaaS) services, such as Service Fabric and Azure Kubernetes Service (AKS), which use virtual machines underneath, can use either availability zones or an availability set to provide application reliability within a region. For more information, see Business continuity with data resiliency.

Considerations for availability

There are several considerations for availability.

Is the application hosted across two or more application platform nodes?

To ensure application platform reliability, it's vital that the application is hosted across at least two nodes. This approach ensures that there are no single points of failure. Ideally an N+1 model should be applied for compute availability where N is the number of instances required to support application availability and performance requirements.

> ⓘ **Note**
>
> Higher SLAs provided for virtual machines and associated related platform services require at least two replica nodes deployed to either an availability set or across two or more availability zones. For more information, see **SLA for Virtual Machines** .

How is the client traffic routed to the application for region, zone, or network outage?

If there's a major outage, client traffic should be routable to application deployments that remain available across other regions or zones. This situation is ultimately where cross-premises connectivity and global load balancing should be used, depending on whether the application is internal or external facing. Services such as Azure Front Door, Azure Traffic Manager, or third-party content delivery networks can route traffic across regions based on application health discovered by using health probes. For more information, see Traffic Manager endpoint monitoring.

Meet data platform requirements

Data and storage services should be running in a highly available configuration or SKU. Azure data platform services offer reliability features to support application reliability. They might only be applicable at a certain SKU. Examples are Azure SQL Database Business Critical SKUs and Azure Storage Zone Redundant Storage (ZRS) with three synchronous replicas spread across availability zones.

Data consistency

Data types should be categorized by data consistency requirements. Data consistency requirements, such as strong or eventual consistency, should be understood for all data types and used to inform data grouping and categorization. Consistency requirements should inform what data replication and synchronization strategies can be considered to meet application reliability targets.

CAP theorem proves that it's impossible for a distributed data store to simultaneously provide more than two guarantees across the following measures:

- *Consistency*. Every read receives the most recent write or an error.
- *Availability*. Every request receives a nonerror response, without the guarantee that it contains the most recent write.
- *Partition tolerance*. A system continues to operate despite an arbitrary number of transactions being dropped or delayed by the network between nodes.

Determining which of these guarantees are most important in the context of application requirements is critical.

Replication and redundancy

Replicating data across zones or paired regions supports application availability objectives to limit the effect of failure scenarios. The ability to restore data from a backup is essential when recovering from data corruption situations and failure scenarios. To ensure sufficient redundancy and availability for zonal and regional failure scenarios, backups should be stored across zones or regions.

Define and test a data restore process to ensure a consistent application state. Regular testing of the data restore process promotes operational excellence and confidence in the ability to recover data in alignment with defined recovery objectives for the application.

Consider how your application traffic is routed to data sources when there's a region, zone, or network outage. Understanding the method used to route application traffic to data sources if there's a major failure event is critical to identify whether failover processes meet recovery objectives. Many Azure data platform services offer native reliability capabilities to handle major failures, such as Azure Cosmos DB Automatic Failover and Azure SQL Database active geo-replication.

> ⓘ **Note**
>
> Some capabilities, such as Azure Storage read-access geo-redundant storage and Azure SQL DB active geo-replication, require application-side failover to alternate endpoints in some failure scenarios. Application logic should be developed to handle these scenarios.

Networking and connectivity requirements

Consider these guidelines to ensure connection availability and improve reliability with Azure services.

Connectivity

- Use a global load balancer to distribute traffic and failover across regions.

  Azure Front Door, Azure Traffic Manager, or third-party content delivery network services can be used to direct inbound requests to external-facing application endpoints deployed across multiple regions. Traffic Manager is a DNS-based load balancer, so failover must wait for DNS propagation to occur. A sufficiently low time-to-live (TTL) value should be used for DNS records, though not all ISPs honor this setting. For application scenarios that require transparent failover, Azure Front Door should be used. For more information, see Disaster Recovery using Azure Traffic Manager and Routing architecture overview.

- For cross-premises connectivity, by using Azure ExpressRoute or VPN, ensure that there are redundant connections from different locations.

  At least two redundant connections should be established across two or more Azure regions and peering locations to ensure there are no single points of failure. An active/active load-shared configuration provides path diversity and promotes availability of network connection paths. For more information, see Cross-network connectivity.

- Simulate a failure path to ensure that connectivity is available over alternative paths.

  The failure of a connection path onto other connection paths should be tested to validate connectivity and operational effectiveness. Using site-to-site VPN connectivity as a backup path for ExpressRoute provides an extra layer of network resiliency for cross-premises connectivity. For more information, see Using site-to-site VPN as a backup for ExpressRoute private peering.

- Eliminate all single points of failure from the data path, both on-premises and hosted on Azure.

  Single-instance Network Virtual Appliances (NVAs) introduce significant connectivity risk, whether deployed in Azure or within an on-premises datacenter. For more information, see Deploy highly available network virtual appliances.

Zone-aware services

- Use ExpressRoute or VPN zone-redundant virtual network gateways.

  Zone-redundant virtual network gateways distribute gateway instances across availability zones to improve reliability and ensure availability during failure scenarios that affect a datacenter in a region. For more information, see About zone-redundant virtual network gateway in Azure availability zones.

- If used, deploy Azure Application Gateway v2 in a zone-redundant configuration.

  Azure Application Gateway v2 can be deployed in a zone-redundant configuration to deploy gateway instances across zones for improved reliability and availability during failure scenarios that affect a datacenter in a region. For more information, see Scaling Application Gateway v2 and WAF v2.

- Use Azure Load Balancer Standard to load-balance traffic across availability zones.

  Azure Load Balancer Standard is zone-aware to distribute traffic across availability zones. It can also be configured in a zone-redundant configuration to improve reliability and ensure availability during failure scenarios that affect a datacenter within a region. For more information, see Load Balancer and availability zones.

- Configure health probes for Azure Load Balancer and Azure Application Gateways.

  Health probes allow Azure Load Balancer to assess the health of backend endpoints to prevent traffic from being sent to unhealthy instances. For more information, see Azure Load Balancer health probes.

- Assess critical application dependencies with health probes.

  Custom health probes should be used to assess overall application health including downstream components and dependent services, such as APIs and datastores. In this approach, traffic isn't sent to backend instances that can't successfully process requests due to dependency failures. For more information, see Health Endpoint Monitoring pattern.

Related links

- To understand business metrics to design resilient Azure applications, see Using business metrics to design resilient Azure applications.
- For information on availability zones, see Build solutions for high availability using availability zones.

- For information on health probes, see Azure Load Balancer health probes and Health Endpoint Monitoring pattern.
- To learn about connectivity risk, see Deploy highly available network virtual appliances.

Next steps

Design reliable Azure applications

Go back to the main article: Design for reliability

Design reliable Azure applications

Article • 05/30/2023

Building a reliable application in the cloud differs from traditional application development. In the past, you might have purchased redundant higher-end hardware to minimize the chance of platform failure. However, for applications that run in the cloud, it's important to acknowledge up front that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. The kinds of failures that you can expect in the cloud are inherent to all highly distributed systems.

Key points

- Use availability zones, where applicable, to improve reliability and optimize costs.
- Design applications to operate when impacted by failures.
- Use the native resiliency capabilities of platform as a service (PaaS) to support overall application reliability.
- Design to scale out and also scale in.
- Validate that required capacity is within Azure service scale limits and quotas.

Use availability zones within a region

Design your application architecture to use availability zones within a region. Availability zones can be used to optimize application availability within a region by providing datacenter-level fault tolerance. However, the application architecture must not share dependencies between zones to use them effectively.

> ⓘ Note
>
> Availability zones might introduce performance and cost considerations for applications that are extremely "chatty" across zones given the implied physical separation between each zone and inter-zone bandwidth charges. For this reason, availability zones can be considered to achieve a higher SLA for lower cost.

Consider if component proximity is required for application performance reasons. If all or part of the application is highly sensitive to latency, components might need to be co-located which can limit the applicability of multi-region and multi-zone strategies.

If your requirements demand even greater failure isolation than availability zones alone can offer, consider deploying to multiple regions. Multiple regions should be used for failover purposes in a disaster state. Other cost needs should be considered. Examples of cost needs are data and networking, and services such as Azure Site Recovery.

Respond to failure

The possibility of failure is unavoidable in public cloud environments. As a result, applications require resilience to respond to outages and deliver reliability. The application should therefore be designed to operate even when impacted by regional, zonal, service, or component failures across critical application scenarios and functionality. Application operations might experience reduced functionality or degraded performance during an outage.

Define an availability strategy to capture how the application remains available during a failure state. The strategy should apply across all application components and the application deployment stamp as a whole, such as via multi-geo scale-unit deployment approach. There are cost implications as well. More resources need to be provisioned in advance to provide high availability. *Active-active* setup, while more expensive than single deployment, can balance cost by lowering load on one stamp and reducing the total amount of resources needed.

In addition to an availability strategy, define a Business Continuity Disaster Recovery (BCDR) strategy for the application and/or its key scenarios. A BCDR strategy should capture how the application responds to a disaster situation, such as a regional outage or the loss of a critical platform service, using either a redeployment, warm-spare *active-passive*, or hot-spare *active-active* approach.

To reduce costs, consider splitting application components and data into groups. For example:

- Must protect
- Nice to protect
- Ephemeral, can be rebuilt or lost instead of protected by the same policy

Considerations to improve reliability

Is the application designed to use managed services?

Azure-managed services provide native resiliency capabilities to support overall application reliability. PaaS offerings should be used to leverage these capabilities. PaaS options are easier to configure and administer. You don't need to provision VMs, set up VNets, manage patches and updates, or do all of the other overhead associated with running software on a VM. To learn more, see Use managed services.

Has the application been designed to scale out?

Azure provides elastic scalability and you should design to scale out. However, applications must leverage a scale-unit approach to navigate service and subscription limits to ensure that individual components and the application as a whole can scale horizontally. Don't forget about scale in, which is important to reduce cost. For example, scale in and out for App Service is done via rules. Often customers write *scale out* rules and never write *scale in* rules, which leaves the App Service more expensive.

Is the application deployed across multiple Azure subscriptions?

Understand the subscription landscape of the application and how components are organized within or across subscriptions when you analyze if relevant subscription limits or quotas can be navigated. Review Azure subscription and service limits to validate that required capacity is within Azure service scale limits and quotas. To learn more, see Azure subscription and service limits.

Related links

- To learn how to minimize dependencies, see Minimize coordination.
- For more information on fault-points and fault-modes, see Failure mode analysis for Azure applications.
- For information on managed services, see Use platform as a service (PaaS) options.

Go back to the main article: Design for reliability

Next steps

Resiliency and dependencies

Resiliency and dependencies

Article • 05/30/2023

To avoid the risk of failure, building failure recovery into the system should be part of the architecture and design phases from the beginning. Dependencies are required for the application to fully operate.

Key points

- Identify possible failure points in the system with failure mode analysis.
- Eliminate all single points of failure.
- Maintain a complete list of application dependencies.
- Ensure that applications can operate in the absence of their dependencies.
- Understand the service-level agreements (SLAs) of individual components in the system to define reliability targets.

Build resiliency with failure mode analysis

Failure mode analysis (FMA) is a process for building resiliency into a system by identifying possible failure points in the system. FMA should be part of the architecture and design phases. Build failure recovery into the system from the beginning.

Fault points describe the elements in an application architecture that are capable of failing. *Fault modes* capture the various ways by which a fault point might fail. Identify all fault points and fault modes. To ensure that an application is resilient to end-to-end failures, it's essential that all fault points and fault modes are understood and operationalized. For more information, see Failure mode analysis for Azure applications.

Eliminate all single points of failure. A single point of failure describes a specific fault point that, if it were to fail, would bring down the entire application. Single points of failure introduce significant risk since any failure of such a component causes an application outage. For more information, see Make all things redundant.

> ⓘ **Note**
>
> Eliminate all *singletons*. A singleton describes a logical component in an application for which there can only be a single instance. This definition can apply to stateful architectural components or application code constructs.

Singletons introduce a significant risk by creating single points of failure in the application design.

Understand the impact of dependencies

Internal dependencies describe components in the application scope that are required for the application to fully operate. *External* dependencies capture required components outside the scope of the application, such as another application or third-party service. Dependencies might be categorized as either strong or weak based on whether the application is able to continue operating in a degraded fashion in their absence. For more information, see Twelve-Factor App: Dependencies.

You should maintain a complete list of application dependencies. Examples of typical dependencies include platform dependencies outside the scope of the application, such as Azure Active Directory, Azure ExpressRoute, or a central Network Virtual Appliance (NVA), and application dependencies such as APIs. For cost purposes, it's important to understand the price for these services and how the services are being charged. For more information, see Develop a cost model.

You can map application dependencies either as a simple list or a document. Usually this decision is part of a design document or reference architecture.

- Understand the effect of an outage with each dependency.

  Strong dependencies play a critical role in application function and availability. Their absence has a significant effect. The absence of weak dependencies might only affect specific features and not affect overall availability. This distinction reflects the cost that is needed to maintain the high availability relationship between the service and its dependencies. Classifying dependencies as either strong or weak helps you identify which components are essential to the application.

- Maintain SLAs and support agreements for critical dependencies.

  A service-level agreement (SLA) represents a commitment around performance and availability of the application. Understanding the SLA of individual components in the system is essential to define reliability targets. Knowing the SLA of dependencies also provides a justification for more spending when making the dependencies highly available and with proper support contracts. The operational commitments of all external and internal dependencies should be understood to inform the broader application operations and health model.

The usage of platform level dependencies, such as Azure Active Directory, must also be understood to ensure that the availability and recovery targets align with the targets of the application.

- Ensure that applications can operate in the absence of their dependencies.

  If the application has strong dependencies that it can't operate without, the availability and recovery targets of these dependencies should align with the targets of the application itself. Make an effort to minimize dependencies to achieve control over application reliability. For more information, see Minimize coordination.

- Ensure that the lifecycle of the application is decoupled from its dependencies.

  If the application lifecycle is closely coupled with the lifecycle of its dependencies, it can limit the operational agility of the application. This fact is true particularly for new releases.

Related links

- For information on fault points and fault modes, see Failure mode analysis for Azure applications.
- For information on single points of failure, see Make all things redundant.
- For information on minimizing dependencies, see Minimize coordination.

Next steps

Best practices for designing reliability

Go back to the main article: Design for reliability

Data management for reliability

Article • 05/30/2023

This article describes database resiliency and database recovery by using geo-restore and active geo-replication. Use the samples described here to understand storage resiliency.

Database resiliency

Azure services offer resiliency, including Azure SQL Database, SQL Server on virtual machines, and Azure Cosmos DB.

Azure SQL Database

SQL Database automatically performs a combination of full database backups weekly, differential database backups hourly, and transaction log backups every five to ten minutes. These backups help protect your business from data loss. Use point-in-time restore to restore a database to an earlier time. For more information, see:

- Restore a database from a backup in Azure SQL Database
- Overview of business continuity with Azure SQL Database

SQL Server on virtual machines

For SQL Server running on virtual machines, there are two options: traditional backups and log shipping. Traditional backups enable you to restore to a specific point in time, but the recovery process is slow. Restoring traditional backups requires starting with an initial full backup, and then applying any backups taken after that backup.

The second option is to configure a log shipping session to delay the restore of log backups, for example, by two hours. This approach provides a window to recover from errors made on the primary.

Azure Cosmos DB

Azure Cosmos DB automatically makes backups at regular intervals. Backups are stored separately in another storage service. Those backups are globally replicated for resiliency against regional disasters. If you accidentally delete your database or collection, you can file a support ticket or call Azure support to restore the data from the last automatic backup. For more information, see Online backup and on-demand data restore in Azure Cosmos DB.

Azure Database for MySQL, Azure Database for PostgreSQL

When you use Azure Database for MySQL or Azure Database for PostgreSQL, the database service automatically makes a backup of the service every five minutes. Using this automatic backup feature, you can restore the server and all its databases into a new server to an earlier point-in-time. For more information, see:

- How to back up and restore a server in Azure Database for MySQL using the Azure portal
- How to backup and restore a server in Azure Database for PostgreSQL using the Azure portal

Distribute data geographically

Azure services support geographically distributed data, such as Azure SQL Database and SQL Server on virtual machines.

SQL Database

Azure SQL Database provides two types of recovery: geo-restore and active geo-replication.

Geo-restore

Geo-restore provides the default recovery option when the database is unavailable because of an incident in the region where your database is hosted. It's also available with Basic, Standard, and Premium databases.

Similar to point-in-time restore, geo-restore relies on database backups in geo-redundant Azure storage. It restores from the geo-replicated backup copy, and therefore is resilient to the storage outages in the primary region. For more information, see Azure SQL Database disaster recovery guidance.

Active geo-replication

Active geo-replication is available for all database tiers. It's designed for applications that have more aggressive recovery requirements than geo-restore can offer. Using active geo-replication, you can create up to four readable secondaries on servers in different regions. You can initiate failover to any of the secondaries.

Active geo-replication can be used to support the application upgrade or relocation scenarios and load balancing for read-only workloads. For more information, see Configure active geo-replication and failover.

For information on how to design and implement applications and applications upgrade without downtime, see Designing globally available services using Azure SQL Database and Managing rolling upgrades of cloud applications by using SQL Database active geo-replication.

SQL Server on Azure Virtual Machines

Various options are available for recovery and high availability for SQL Server 2012 and later that run in Azure Virtual Machines. For more information, see Business continuity and HADR for SQL Server on Azure Virtual Machines.

SQL Server Always On availability groups across regions

You can use SQL Always On availability groups for high availability by creating a single availability group that includes the SQL Server instances in both regions.

As an example, the Multi-region N-tier application reference architecture shows practices for running an N-tier application in multiple Azure regions to achieve availability and a robust disaster recovery infrastructure. It uses a SQL Server Always On availability group and Azure Traffic Manager.

Storage resiliency

Azure Storage provides data resiliency through automated replicas. However, this ability doesn't prevent application code or users from corrupting data, whether accidentally or maliciously. Maintaining data fidelity in the face of application or user error requires more advanced techniques, such as copying the data to a secondary storage location with an audit log.

- *Block blobs*. Create a point-in-time snapshot of each block blob. For more information, see Create a snapshot of a blob.

For each snapshot, you're only charged for the storage required to store the differences within the blob since the last snapshot state. The snapshots are dependent on the existence of the original blob that they're based on. We recommend a copy operation to another blob or even another storage account. This approach ensures that backup data is properly protected against accidental deletion.

You can use AzCopy or Azure PowerShell to copy the blobs to another storage account.

- *Files*. Use share snapshots, or use AzCopy or PowerShell, to copy your files to another storage account.

- *Tables*. Use AzCopy to export the table data into another storage account in another region.

For samples related to storage resiliency, see Storage resiliency code samples. The scripts demonstrate these tasks:

- Deploy storage accounts, blob containers, and a file share with an Azure Resource Manager template.
- Copy a local file into a blob container.
- Create a blob snapshot.
- Copy a local file into a file share.
- Create a share snapshot.
- Use AzCopy to copy a blob container from one storage account to another.

Design reliable Azure applications

Article • 05/30/2023

Building a reliable application in the cloud differs from traditional application development. In the past, you might have purchased redundant higher-end hardware to minimize the chance of platform failure. However, for applications that run in the cloud, it's important to acknowledge up front that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. The kinds of failures that you can expect in the cloud are inherent to all highly distributed systems.

Key points

- Use availability zones, where applicable, to improve reliability and optimize costs.
- Design applications to operate when impacted by failures.
- Use the native resiliency capabilities of platform as a service (PaaS) to support overall application reliability.
- Design to scale out and also scale in.
- Validate that required capacity is within Azure service scale limits and quotas.

Use availability zones within a region

Design your application architecture to use availability zones within a region. Availability zones can be used to optimize application availability within a region by providing datacenter-level fault tolerance. However, the application architecture must not share dependencies between zones to use them effectively.

> ⓘ **Note**
>
> Availability zones might introduce performance and cost considerations for applications that are extremely "chatty" across zones given the implied physical separation between each zone and inter-zone bandwidth charges. For this reason, availability zones can be considered to achieve a higher SLA for lower cost.

Consider if component proximity is required for application performance reasons. If all or part of the application is highly sensitive to latency, components might need to be co-located which can limit the applicability of multi-region and multi-zone strategies.

If your requirements demand even greater failure isolation than availability zones alone can offer, consider deploying to multiple regions. Multiple regions should be used for failover purposes in a disaster state. Other cost needs should be considered. Examples of cost needs are data and networking, and services such as Azure Site Recovery.

Respond to failure

The possibility of failure is unavoidable in public cloud environments. As a result, applications require resilience to respond to outages and deliver reliability. The application should therefore be designed to operate even when impacted by regional, zonal, service, or component failures across critical application scenarios and functionality. Application operations might experience reduced functionality or degraded performance during an outage.

Define an availability strategy to capture how the application remains available during a failure state. The strategy should apply across all application components and the application deployment stamp as a whole, such as via multi-geo scale-unit deployment approach. There are cost implications as well. More resources need to be provisioned in advance to provide high availability. *Active-active* setup, while more expensive than single deployment, can balance cost by lowering load on one stamp and reducing the total amount of resources needed.

In addition to an availability strategy, define a Business Continuity Disaster Recovery (BCDR) strategy for the application and/or its key scenarios. A BCDR strategy should capture how the application responds to a disaster situation, such as a regional outage or the loss of a critical platform service, using either a redeployment, warm-spare *active-passive*, or hot-spare *active-active* approach.

To reduce costs, consider splitting application components and data into groups. For example:

- Must protect
- Nice to protect
- Ephemeral, can be rebuilt or lost instead of protected by the same policy

Considerations to improve reliability

Is the application designed to use managed services?

Azure-managed services provide native resiliency capabilities to support overall application reliability. PaaS offerings should be used to leverage these capabilities. PaaS options are easier to configure and administer. You don't need to provision VMs, set up VNets, manage patches and updates, or do all of the other overhead associated with running software on a VM. To learn more, see Use managed services.

Has the application been designed to scale out?

Azure provides elastic scalability and you should design to scale out. However, applications must leverage a scale-unit approach to navigate service and subscription limits to ensure that individual components and the application as a whole can scale horizontally. Don't forget about scale in, which is important to reduce cost. For example, scale in and out for App Service is done via rules. Often customers write *scale out* rules and never write *scale in* rules, which leaves the App Service more expensive.

Is the application deployed across multiple Azure subscriptions?

Understand the subscription landscape of the application and how components are organized within or across subscriptions when you analyze if relevant subscription limits or quotas can be navigated. Review Azure subscription and service limits to validate that required capacity is within Azure service scale limits and quotas. To learn more, see Azure subscription and service limits.

Related links

- To learn how to minimize dependencies, see Minimize coordination.
- For more information on fault-points and fault-modes, see Failure mode analysis for Azure applications.
- For information on managed services, see Use platform as a service (PaaS) options.

Go back to the main article: Design for reliability

Next steps

Resiliency and dependencies

Use business metrics to design resilient Azure applications

Article • 05/30/2023

You can use business metrics to design resilient applications in Azure. You need to understand workload availability targets, recovery metrics, and availability metrics.

Workload availability targets

Are availability targets, such as service-level agreements (SLAs), service-level indicators (SLIs), and service-level objectives (SLOs), defined for the application or key scenarios?

Understanding customer availability expectations is vital to reviewing overall operations for the application. For instance, if a customer wants an application SLO of `99.999%`, the level of inherent operational activity required by the application is going to be far greater than if an SLO of `99.9%` is the goal.

Define your own target SLAs for each workload in your solution so you can determine whether the architecture meets the business requirements.

Consider cost and complexity

Everything else being equal, higher availability is better. But as you strive for more nines, the cost and complexity grow. An uptime of `99.99%` translates to about five minutes of total downtime per month. Is it worth the extra complexity and cost to reach five nines? The answer depends on your business requirements.

Here are some other considerations when defining an SLA:

- To achieve *four nines* (`99.99%`), you can't rely on manual intervention to recover from failures. The application must be self-diagnosing and self-healing.
- Beyond four nines, it's challenging to detect outages quickly enough to meet the SLA.
- Think about the time window that your SLA is measured against. The smaller the window, the tighter the tolerances. It doesn't make sense to define your SLA in terms of hourly or daily uptime.
- Consider the *mean time to recover* (MTTR) and *mean time between failures* (MTBF) measurements. The higher your SLA, the less frequently the service can go down and the quicker the service must recover.

- Get agreement from your customers for the availability targets of each piece of your application, and document it. Otherwise, your design might not meet your customers' expectations.

Identify dependencies

Perform dependency-mapping exercises to identify internal and external dependencies. Examples include dependencies relating to security or identity, such as Active Directory or third-party services. Third-party services might include a payment provider or email messaging service.

Pay particular attention to external dependencies that might be a single point of failure or cause bottlenecks. If a workload requires `99.99%` uptime but depends on a service with a `99.9%` SLA, that service can't be a single point of failure in the system. One remedy is to have a fallback path in case the service fails. Alternatively, take other measures to recover from a failure in that service.

The following table shows the potential cumulative downtime for various SLA levels.

SLA	Downtime per week	Downtime per month	Downtime per year
`99%`	`1.68` hours	`7.2` hours	`3.65` days
`99.9%`	`10.1` minutes	`43.2` minutes	`8.76` hours
`99.95%`	`5` minutes	`21.6` minutes	`4.38` hours
`99.99%`	`1.01` minutes	`4.32` minutes	`52.56` minutes
`99.999%`	`6` seconds	`25.9` seconds	`5.26` minutes

Identify critical system flows

Understanding critical system flows is vital to assessing overall operational effectiveness and should be used to inform a health model for the application. It can also tell if areas of the application are over or underutilized and should be adjusted to better meet business needs and cost goals.

Critical subsystems or paths through the application might have higher expectations around availability, recovery, and performance due to the criticality of associated business scenarios and functionality. This fact also helps to understand if cost might be affected due to these higher needs.

Identify less critical components

Some less critical components or paths through the application might have lower expectations around availability, recovery, and performance. Less critical components can result in cost reduction by choosing lower SKUs with less performance and availability.

Recovery metrics

Derive these values by conducting a risk assessment, and make sure that you understand the cost and risk of downtime and data loss. These nonfunctional requirements of a system should be dictated by business requirements.

- *Recovery time objective* (RTO) is the maximum acceptable time an application is unavailable after an incident.
- *Recovery point objective* (RPO) is the maximum duration of data loss that's acceptable during a disaster.

If the MTTR value of *any* critical component in a highly available setup exceeds the system RTO, a failure in the system might cause an unacceptable business disruption. That is, you can't restore the system within the defined RTO.

Availability metrics

Use these measures to plan for redundancy and determine customer SLAs.

- *Mean time to recover* (MTTR) is the average time it takes to restore a component after a failure.
- *Mean time between failures* (MTBF) is how long a component can reasonably expect to last between outages.

Understand service-level agreements

In Azure, the Service Level Agreement describes Microsoft's commitments for uptime and connectivity. If the SLA for a particular service is `99.9%`, you should expect the service to be available `99.9%` of the time. Different services have different SLAs.

The Azure SLA also includes procedures for obtaining a service credit if the SLA isn't met, along with specific definitions of *availability* for each service. That aspect of the SLA acts as an enforcement policy.

The [Service Level Agreement Estimator] sample shows how to calculate the SLA of your architecture.

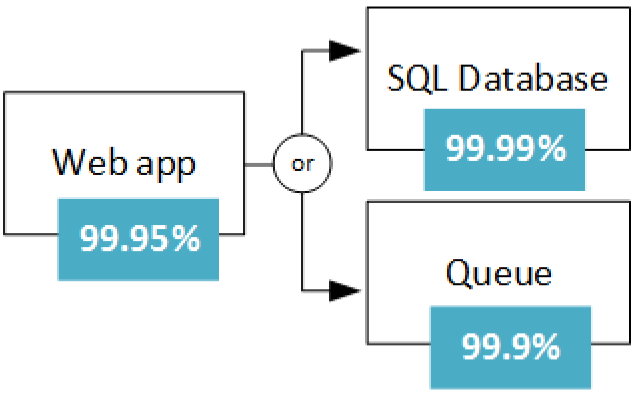
Composite SLAs

Composite SLAs involve multiple services supporting an application, with differing levels of availability. For example, consider an App Service web app that writes to Azure SQL Database. Currently, these Azure services have the following SLAs:

- App Service web apps = `99.95%`
- SQL Database = `99.99%`

What is the maximum downtime you would expect for this application? If either service fails, the whole application fails. The probability of each service failing is independent, so the composite SLA for this application is `99.95% × 99.99% = 99.94%`. That value is lower than the individual SLAs, which isn't surprising because an application that relies on multiple services has more potential failure points.

You can improve the composite SLA by creating independent fallback paths. For example, if SQL Database is unavailable, put transactions into a queue to be processed later:



With this design, the application is still available even if it can't connect to the database. However, it fails if the database and the queue both fail at the same time. The expected percentage of time for a simultaneous failure is `0.0001 × 0.001`, so the composite SLA for this combined path is:

- Database *or* queue = `1.0 − (0.0001 × 0.001) = 99.99999%`

The total composite SLA is:

- Web app *and* (database *or* queue) = `99.95% × 99.99999% = ~99.95%`

There are tradeoffs to this approach. The application logic is more complex, you're paying for the queue, and you need to consider data consistency issues.

SLAs for multiregion deployments

SLAs for multiregion deployments involve a high-availability technique to deploy the application in more than one region and use Azure Traffic Manager to fail over if the application fails in one region.

The composite SLA for a multiregion deployment is calculated as follows:

- *N* is the composite SLA for the application deployed in one region.
- *R* is the number of regions where the application is deployed.

The expected chance that the application fails in all regions at the same time is (`(1 - N) ^ R`). For example, if the single-region SLA is `99.95%`:

- The combined SLA for two regions = `(1 - (1 - 0.9995) ^ 2) = 99.999975%`
- The combined SLA for four regions = `(1 - (1 - 0.9995) ^ 4) = 99.999999%`

The SLA for Traffic Manager is also a factor. Failing over isn't instantaneous in active-passive configurations, which can result in downtime during a failover. See Traffic Manager endpoint monitoring.

Best practices for designing reliability in Azure applications

Article • 05/30/2023

This article describes best practices to design Azure applications for reliability. These best practices are derived from our experience with Azure reliability and the experiences of customers like yourself.

During the architectural phase, implement practices that meet your business requirements, identify failure points, and minimize the scope of failures.

Build availability targets and recovery targets into your design

A service-level agreement (SLA) is an availability target that represents a commitment around performance and availability of the application. Understanding the SLA of individual components within the system is essential in order to define reliability targets. Recovery targets identify how long the workload can be unavailable and how much data is acceptable to lose during a disaster.

Define target reports for the application and key scenarios. There might be penalties, such as financial charges, associated with failing to meet SLA commitments. The consequences of not satisfying availability targets should be fully understood.

Ensure the application and data platforms meet your reliability requirements

Designing application platform and data platform resiliency and availability are critical to ensure overall application reliability.

Ensure connectivity

To ensure connection availability and improve reliability with Azure services:

- Use a global load balancer to distribute traffic and failover across regions.
- For cross-premises connectivity with Azure ExpressRoute or VPN, ensure that there are redundant connections from different locations.

- Simulate a failure path to ensure that connectivity is available over alternative paths.
- Eliminate all single points of failure from the data path, both on-premises and hosted by Azure.

Use zone-aware services

Zone-aware services can improve reliability and ensure availability during failure scenarios that affect a datacenter within a region. They can also be used to deploy gateway instances across zones for improved reliability and availability during failure scenarios that affect a datacenter within a region.

Design resilience to respond to outages

Applications should be designed to operate even when affected by regional, zonal, service, or component failures across critical application scenarios and functionality. Application operations might experience reduced functionality or degraded performance during an outage.

Perform a failure mode analysis

Failure mode analysis (FMA) builds resiliency into an application early in the design stage. It helps you identify the types of failures that your application might experience, the potential effects of each, and possible recovery strategies.

Have all single points of failure been eliminated? A single point of failure describes a specific fault-point that, if it were to fail, would bring down the entire application. Single points of failure introduce significant risk since any failure of this component causes an application outage.

Have all fault-points and fault-modes been identified? Fault-points describe the elements within an application architecture that are capable of failing, while fault-modes capture the various ways by which a fault-point might fail. To ensure that an application is resilient to end-to-end failures, it's essential that all fault-points and fault-modes are understood and operationalized.

Understand the effect of an outage with each dependency

Strong dependencies play a critical role in application function and availability. Their absence has a significant effect, while the absence of weak dependencies might only affect specific features and not affect overall availability. Dependencies can be categorized as either strong or weak based on whether the application is able to continue operating in a degraded fashion in their absence.

Design for scalability

A cloud application must be able to scale to accommodate changes in usage. Begin with discrete components and design the application to respond automatically to load changes whenever possible. Keep scaling limits in mind during design so you can expand easily in the future.

Next steps

Testing for reliability

Go back to the main article: Design for reliability

Checklist for reliability testing

Article • 05/30/2023

Regular testing should be performed as part of each major change and, if possible, on a regular basis to validate existing thresholds, targets, and assumptions. Testing should also ensure the validity of the health model, capacity model, and operational procedures.

Checklist

Have you tested your applications with reliability in mind?

- ✓ Test regularly to validate existing thresholds, targets, and assumptions.
- ✓ Automate testing as much as possible.
- ✓ Perform testing on both key test environments and the production environment.
- ✓ Perform chaos testing by injecting faults.
- ✓ Create and test a disaster recovery plan on a regular basis by using key failure scenarios.
- ✓ Design a disaster recovery strategy to run most applications with reduced functionality.
- ✓ Design a backup strategy that's tailored for the business requirements and circumstances of the application.
- ✓ Test and validate the failover and failback approach successfully at least once.
- ✓ Configure request timeouts to manage intercomponent calls.
- ✓ Implement retry logic to handle transient application failures and transient failures with internal or external dependencies.
- ✓ Configure and test health probes for your load balancers and traffic managers.
- ✓ Apply chaos principles continuously.
- ✓ Create and organize a central chaos engineering team.

Azure services

- Azure Site Recovery
- Azure Pipelines
- Azure Traffic Manager
- Azure Load Balancer

Reference architecture

- Failure mode analysis for Azure applications
- High availability and disaster recovery scenarios for IaaS apps
- Back up files and applications on Azure Stack Hub

Related links

- For information on performance testing, see Performance testing.
- For information on chaos engineering, see Chaos engineering.
- For information on failure and disaster recovery, see Backup and disaster recovery for Azure applications.
- For information on testing applications, see Testing your application and Azure environment.

Next steps

Resiliency testing

Testing applications for availability and resiliency

Article • 05/30/2023

Applications should be tested to ensure *availability* and *resiliency*. Availability describes the amount of time that an application runs in a healthy state without significant downtime. Resiliency describes how quickly an application recovers from failure.

Being able to measure availability and resiliency can answer questions like: How much downtime is acceptable? How much does potential downtime cost your business? What are your availability requirements? How much do you invest in making your application highly available? What is the risk versus the cost? Testing plays a critical role in making sure your applications can meet these requirements.

Key points

- Test regularly to validate existing thresholds, targets, and assumptions.
- Automate testing as much as possible.
- Perform testing on both key test environments and the production environment.
- Verify how the end-to-end workload performs under intermittent failure conditions.
- Test the application against critical functional and nonfunctional requirements for performance.
- Conduct load testing with expected peak volumes to test scalability and performance under load.
- Perform chaos testing by injecting faults.

When to test

Regular testing should be performed as part of each major change and, if possible, on a regular basis to validate existing thresholds, targets, and assumptions. While most testing should be performed within the testing and staging environments, it's often beneficial to also run a subset of tests against the production system. Plan a 1:1 parity of key test environments with the production environment.

> ⓘ Note
>
> Automate testing where possible to ensure consistent test coverage and reproducibility. Automate common testing tasks and integrate them into your build

processes. Manually testing software is tedious and susceptible to error, although manual explorative testing can also be conducted.

Testing for resiliency

To test resiliency, you should verify how the end-to-end workload performs under intermittent failure conditions.

Run tests in production using both synthetic and real user data. Test and production are rarely identical, so it's important to validate your application in production using a blue-green or canary deployment. This way, you're testing the application under real conditions, so you can be sure that it functions as expected when fully deployed.

As part of your test plan, include:

- Chaos engineering
- Automated pre-deployment testing
- Fault injection testing
- Peak load testing
- Disaster recovery testing

Performance testing

The primary goal of performance testing is to validate benchmark behavior for the application. Performance testing is the superset of both *load testing* and *stress testing*.

Load testing validates application scalability by rapidly and/or gradually increasing the load on the application until it reaches a threshold or limit. Stress testing involves various activities to overload existing resources and remove components to understand overall resiliency and how the application responds to issues.

Simulation testing

Simulation testing involves creating small, real-life situations. Simulations demonstrate the effectiveness of the solutions in the recovery plan and highlight any issues that weren't adequately addressed.

As you perform simulation testing, follow best practices:

- Conduct simulations in a manner that doesn't disrupt actual business but feels like a real situation.

- Make sure that simulated scenarios are completely controllable. If the recovery plan seems to be failing, you can restore the situation back to normal without causing damage.
- Inform management about when and how the simulation exercises are conducted. Your plan should detail the time frame and the resources affected during the simulation.

Fault injection testing

For fault injection testing, check the resiliency of the system during failures, either by triggering actual failures or by simulating them. Here are some strategies to induce failures:

- Shut down virtual machine (VM) instances.
- Crash processes.
- Expire certificates.
- Change access keys.
- Shut down the DNS service on domain controllers.
- Limit available system resources, such as RAM or number of threads.
- Unmount disks.
- Redeploy a VM.

Your test plan should incorporate possible failure points identified during the design phase, in addition to common failure scenarios:

- Test your application in an environment as close to production as possible.
- Test failures in combination.
- Measure the recovery times, and be sure that your business requirements are met.
- Verify that failures don't cascade and are handled in an isolated way.

Test under peak loads

Load testing is crucial to identify failures that only happen under load, such as the backend database being overwhelmed or service throttling. Test for peak load and anticipated increase in peak load, using production data or synthetic data that's as close to production data as possible. Your goal is to see how the application behaves under real-world conditions.

Related links

- For more test types, see Test types.

- To learn about load and stress tests, see Performance testing.
- To learn about chaos testing, see Chaos engineering.

Go back to the main article: Checklist for reliability testing

Next steps

Backup and disaster recovery

Backup and disaster recovery for Azure applications

Article • 05/30/2023

Disaster recovery is the process of restoring application functionality after a catastrophic loss.

In cloud environments, we acknowledge up front that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Testing is one way to minimize these effects. You should automate testing of your applications where possible, but you also need to be prepared for when they fail. When a failure happens, having backup and recovery strategies becomes important.

Your tolerance for reduced functionality during a disaster is a business decision that varies from one application to the next. It might be acceptable for some applications to be temporarily unavailable, or partially available with reduced functionality or delayed processing. For other applications, any reduced functionality is unacceptable.

Key points

- Create and test a disaster recovery plan regularly using key failure scenarios.
- Design a disaster recovery strategy to run most applications with reduced functionality.
- Design a backup strategy that's tailored for the business requirements and circumstances of the application.
- Automate failover and failback steps and processes.
- Test and validate the failover and failback approach successfully at least once.

Disaster recovery plan

Start by creating a recovery plan. The plan is considered complete after it's been fully tested. Include the people, processes, and applications needed to restore functionality within the service-level agreement (SLA) that you've defined for your customers.

Consider the following suggestions when you create and test your disaster recovery plan:

- Describe how to contact support and how to escalate issues. This information helps to avoid prolonged downtime as you work out the recovery process for the first time.

- Evaluate the business impact of application failures.
- Choose a cross-region recovery architecture for mission-critical applications.
- Identify a specific owner of the disaster recovery plan, including automation and testing.
- Document the process, especially any manual steps.
- Automate the process as much as possible.
- Establish a backup strategy for all reference and transactional data, and test backup restoration regularly.
- Set up alerts for the stack of Azure services consumed by your application.
- Train operations staff to execute the plan.
- Perform regular disaster simulations to validate and improve the plan.

If you use Azure Site Recovery to replicate virtual machines (VMs), create a fully automated recovery plan to failover the entire application.

Operational readiness testing

Perform an operational readiness test for failover to the secondary region and for failback to the primary region. Many Azure services support manual failover or test failover for disaster recovery drills. You can simulate an outage by shutting down or removing Azure services.

Automated operational responses should be tested frequently as part of the normal application lifecycle to ensure operational effectiveness.

Failover and failback testing

Test failover and failback to verify that your application's dependent services come back up in a synchronized manner during disaster recovery. Changes to systems and operations might affect failover and failback functions, but the impact might not be detected until the main system fails or becomes overloaded. Test failover capabilities *before* they're required to compensate for a live problem. Also, be sure dependent services failover and failback in the correct order.

If you use Azure Site Recovery to replicate VMs, run disaster recovery drills periodically by testing failovers to validate your replication strategy. A test failover doesn't affect the ongoing VM replication or your production environment. For more information, see Run a disaster recovery drill to Azure.

Dependent service outage

For each dependent service, you should understand the implications of service disruption and how the application responds. Many services include features that support resiliency and availability, so evaluating each service independently is likely to improve your disaster recovery plan. For example, Azure Event Hubs supports failing over to the secondary namespace.

Network outage

When parts of the Azure network are inaccessible, you might not be able to access your application or data. For this situation, you should design the disaster recovery strategy to run most applications with reduced functionality.

If reducing functionality isn't an option, the remaining options are application downtime or failover to an alternate region.

In a reduced functionality scenario:

- If your application can't access its data because of an Azure network outage, you can run locally with reduced application functionality by using cached data.
- You can store data in an alternate location until connectivity is restored.

Recovery automation

The steps required to recover or failover the application to a secondary Azure region in failure situations should be codified, preferably in an automated manner, to ensure capabilities exist to respond effectively to an outage in a way that limits impact. Similar codified steps should also exist to capture the process required to failback the application to the primary region once a failover triggering issue has been addressed.

When automating failover procedures, ensure that the tooling used for orchestrating the failover is also considered in the failover strategy. For example, if you run your failover from Jenkins running on a VM, you'll be in trouble if that virtual machine is part of the outage. Azure DevOps Projects are scoped to a region too.

Backup strategy

Many alternative strategies are available for implementing distributed compute across regions. These strategies must be tailored to the specific business requirements and circumstances of the application. At a high level, the approaches can be divided into the following categories:

- **Redeploy on disaster**: In this approach, the application is redeployed from scratch at the time of disaster. Redeploying from scratch is appropriate for non-critical applications that don't require a guaranteed recovery time.

- **Warm spare (Active/Passive)**: Create a secondary hosted service in an alternate region, and deploy roles to guarantee minimal capacity. However, these roles don't receive production traffic. This approach is useful for applications that haven't been designed to distribute traffic across regions.

- **Hot spare (Active/Active)**: The application is designed to receive production load in multiple regions. The cloud services in each region might be configured for higher capacity than required for disaster recovery purposes. Instead, the cloud services might scale out as necessary at the time of a disaster and failover. This approach requires a large investment in application design, but it has significant benefits. These include low and guaranteed recovery time, continuous testing of all recovery locations, and efficient usage of capacity.

Plan for regional failures

Azure is divided physically and logically into units called regions. A region consists of one or more datacenters in close proximity. Many regions and services also support availability zones, which can be used to provide more resiliency against outages in a single datacenter. Consider using regions with availability zones to improve the availability of your solution.

Under rare circumstances, it's possible that facilities in an entire availability zone or region can become inaccessible, for example, because of network failures. Or, facilities can be lost entirely, for example, because of a natural disaster. Azure has capabilities for creating applications that are distributed across zones and regions. Such distribution helps to minimize the possibility that a failure in one zone or region could affect other zones or regions.

Related links

- For information on testing failovers, see Run a disaster recovery drill to Azure.
- For information on Event Hubs, see Azure Event Hubs.

Go back to the main article: Testing for reliability

Next step

Automatic retry of failed backup jobs

Automatic retry of failed backup jobs

Article • 05/30/2023

Azure Backup is a simple, secure, and cost-effective solution that comprehensively protects your data assets in Azure and requires zero infrastructure. Azure's built-in data protection covers a wide range of workloads, and helps protect your mission-critical workloads running in the cloud. Azure ensures that your backups are always available and managed at scale across your entire backup estate.

As a backup user or administrator, you can monitor all backup solutions and configure alerts to notify you about important events.

Many failures or outages are transient. You can solve them just by retrying the backup, or the restore job. However, waiting for an engineer to retry the job manually or assign the relevant permission wastes valuable time. Automation is the smarter way to retry failed jobs, and ensures that you continue to meet your target Recovery Point Objectives (RPOs) with one successful backup per day.

Retrieve relevant backup data through Azure Resource Graph (ARG) and combine the data with corrective PowerShell and CLI steps. This article explains how you can retry backups for all failed jobs using ARG and PowerShell.

Prerequisites

You need an Azure Automation account. You can use an existing account or create a new account with one user-assigned managed identity, at minimum.

Add modules

After you create the automation account, install the following modules by navigating to the individual module gallery:

- Az.Accounts
- Az.RecoveryServices
- Az.Resources
- Az.ManagedServiceIdentity
- Az.Graph

Assign permissions to managed identities

To assign permissions to managed identities, complete the following steps:

1. Sign in to Azure interactively using the Connect-AzAccount cmdlet and follow the instructions:

Azure PowerShell

```
# Sign in to your Azure subscription

$sub = Get-AzSubscription -ErrorAction SilentlyContinue
if(-not($sub))
{
 Connect-AzAccount
}

# If you have multiple subscriptions, set the one to use
# Select-AzSubscription -SubscriptionId <SUBSCRIPTIONID>
```

2. Provide an appropriate value for the following variables, and then run the script:

Azure PowerShell

```
$resourceGroup = "resourceGroupName"

# These values are used in this tutorial
$automationAccount = "xAutomationAccount"
$userAssignedManagedIdentity = "xUAMI"
```

3. Use the PowerShell cmdlet New-AzRoleAssignment to assign a role to the *system-assigned* managed identity:

PowerShell

```
$role1 = "DevTest Labs User"

$SAMI = (Get-AzAutomationAccount -ResourceGroupName $resourceGroup
-Name $automationAccount).Identity.PrincipalId
New-AzRoleAssignment
 -ObjectId $SAMI
 -ResourceGroupName $resourceGroup
 -RoleDefinitionName $role1
```

4. You need the same role assignment for the *user-assigned* managed identity:

PowerShell

```
$UAMI = (Get-AzUserAssignedIdentity -ResourceGroupName
$resourceGroup -Name $userAssignedManagedIdentity).PrincipalId
```

```
New-AzRoleAssignment
    -ObjectId $UAMI
    -ResourceGroupName $resourceGroup
    -RoleDefinitionName $role1
```

5. You need extra permissions for the *system-assigned* managed identity to run the cmdlets `Get-AzUserAssignedIdentity` and `Get-AzAutomationAccount`:

PowerShell

```powershell
$role2 = "Reader"
New-AzRoleAssignment
 -ObjectId $SAMI
 -ResourceGroupName $resourceGroup
 -RoleDefinitionName $role2
```

Create a PowerShell runbook

To create a runbook that can be run by managed identities, complete the following steps:

1. Sign in to the Azure portal and navigate to your Automation account.

2. Under **Process Automation** in the side panel, select **Runbooks**.

3. Select **Create a runbook**:
   a. Name the runbook *miTesting*.
   b. From the **Runbook type** dropdown menu, select **PowerShell**.
   c. Select **Create**.

4. In the runbook editor, paste the following code:

PowerShell

```powershell
$connection = Get-AutomationConnection -Name AzureRunAsConnection

$connectionResult = Connect-AzAccount
-ServicePrincipal
-Tenant $connection.TenantID
-ApplicationId $connection.ApplicationID
-CertificateThumbprint $connection.CertificateThumbprint
"Login successful.."

$query = "RecoveryServicesResources
| where type in~ ('microsoft.recoveryservices/vaults/backupjobs')
| extend vaultName = case(type =~
'microsoft.dataprotection/backupVaults/backupJobs',
```

```
properties.vaultName,type =~
'Microsoft.RecoveryServices/vaults/backupJobs', split(split(id,
'/Microsoft.RecoveryServices/vaults/')[1],'/')[0],'--')
| extend friendlyName = case(type =~
'microsoft.dataprotection/backupVaults/backupJobs',
strcat(properties.dataSourceSetName , '/', properties.dataSourceName),
type =~ 'Microsoft.RecoveryServices/vaults/backupJobs',
properties.entityFriendlyName, '--')
| extend dataSourceType = case(type =~
'Microsoft.RecoveryServices/vaults/backupJobs',
properties.backupManagementType, type =~
'microsoft.dataprotection/backupVaults/backupJobs',
properties.dataSourceType, '--')
| extend protectedItemName = split(split(properties.backupInstanceId,
'protectedItems')[1],'/')[1]
| extend vaultId = tostring(split(id, '/backupJobs')[0])
| extend vaultSub = tostring( split(id, '/')[2])
| extend jobStatus = case (properties.status == 'Completed' or
properties.status ==
'CompletedWithWarnings','Succeeded',properties.status == 'Failed',
'Failed', properties.status == 'InProgress', 'Started',
properties.status), operation = case(type =~
'microsoft.dataprotection/backupVaults/backupJobs' and
tolower(properties.operationCategory) =~ 'backup' and
properties.isUserTriggered == 'true',
strcat('adhoc',properties.operationCategory), type =~
'microsoft.dataprotection/backupVaults/backupJobs',
tolower(properties.operationCategory), type =~
'Microsoft.RecoveryServices/vaults/backupJobs' and
tolower(properties.operation) =~ 'backup' and
properties.isUserTriggered == 'true', strcat('adhoc',
properties.operation), type =~
'Microsoft.RecoveryServices/vaults/backupJobs',
tolower(properties.operation), '--'), startTime =
todatetime(properties.startTime), endTime = properties.endTime,
duration = properties.duration
| where startTime >= ago(24h)
| where (dataSourceType in~ ('AzureIaasVM'))
| where jobStatus=='Failed'
| where operation == 'backup' or operation == 'adhocBackup'
| project vaultSub, vaultId, protectedItemName, startTime, endTime,
jobStatus, operation
| sort by vaultSub"

$subscriptions = Get-AzSubscription | foreach {$_.SubscriptionId}

$result = Search-AzGraph -Subscription $subscriptions -Query $query -
First 5

$result = $result.data


$prevsub = ""
foreach($jobresponse in $result)
{
```

```
                if($jobresponse.vaultSub -ne $prevsub)
            {
                        Set-AzContext -SubscriptionId
$jobresponse.vaultSub
                        $prevsub = $jobresponse.vaultSub
            }
            $item = Get-AzRecoveryServicesBackupItem -VaultId
$jobresponse.vaultId -BackupManagementType AzureVM -WorkloadType
AzureVM -Name
$jobresponse.protectedItemName

            Backup-AzRecoveryServicesBackupItem -ExpiryDateTimeUTC
(get-date).AddDays(10) -Item $item -VaultId $jobresponse.vaultId
}
```

5. Select **Save** and then **Test pane**.

You successfully created a PowerShell runbook.

Create a new schedule with PowerShell

To create a new schedule with PowerShell:

- Use the New-AzAutomationSchedule cmdlet to create schedules.
- Specify the start time for the schedule and the frequency it should run.

The following code example shows how to create a recurring schedule that runs every day at *1:00 PM* for one year:

PowerShell

```
PS C:\> $StartTime = Get-Date "13:00:00"
PS C:\> $EndTime = $StartTime.AddYears(1)
PS C:\> New-AzAutomationSchedule -AutomationAccountName
"MyAutomationAccount" -Name "Schedule02" -StartTime $StartTime -ExpiryTime
$EndTime -DayInterval 1 -ResourceGroupName "ResourceGroup01"
```

The first command creates a date object using the `Get-Date` cmdlet and then stores the object in the `$StartDate` variable. The specified time must be, at least, five minutes in the future.

The second command creates a date object using the `Get-Date` cmdlet and then stores the object in the `$EndDate` variable. The command specifies a future time.

The final command creates a daily schedule named *Schedule02* to begin at the time stored in `$StartDate` and expires at the time stored in `$EndDate`.

Link a schedule to a runbook

Consider the following concepts when you link a schedule to a runbook:

- You can link a runbook to multiple schedules, and a schedule can have multiple runbooks linked to it.
- If a runbook has parameters, you can provide values for them.
- Provide values for any mandatory parameters and any optional parameters. These values are used each time the runbook is started by this schedule.
- You can attach the same runbook to another schedule and specify different parameter values.

Link a schedule to a runbook by using PowerShell

To link a schedule to a runbook by using PowerShell:

- Use the Register-AzAutomationScheduledRunbook cmdlet to link a schedule.
- You can specify parameter values for the runbook with the Parameters parameter.

For more information about how to specify parameter values, see Start a runbook in Azure Automation.

The following code example shows how to link a schedule to a runbook by using an Azure Resource Manager cmdlet with parameters:

PowerShell

```
$automationAccountName = "MyAutomationAccount"
$runbookName = "Test-Runbook"
$scheduleName = "Sample-DailySchedule"
$params =
@{"FirstName"="Joe";"LastName"="Smith";"RepeatCount"=2;"Show"=$true}
Register-AzAutomationScheduledRunbook -AutomationAccountName
$automationAccountName
-Name $runbookName -ScheduleName $scheduleName -Parameters $params `
-ResourceGroupName "ResourceGroup01"
```

Next steps

Error handling

Error handling for resilient applications in Azure

Article • 05/30/2023

In a distributed system, ensuring that your application can recover from errors is critical. You can test your applications to prevent errors and failure, but you need to prepare for a wide range of issues. Testing doesn't always catch everything, so you should understand how to handle errors and prevent potential failure.

Many things in a distributed system, such as underlying cloud infrastructure and third-party runtime dependencies, are outside your span of control and your means to test. You can be sure something will fail eventually, so you need to be prepared.

Key points

- Implement retry logic to handle transient application failures and transient failures with internal or external dependencies.
- Uncover issues or failures in your application's retry logic.
- Configure request timeouts to manage intercomponent calls.
- Configure and test health probes for your load balancers and traffic managers.
- Segregate read operations from update operations across application data stores.

Implement retry logic

A retry pattern describes how an application handles anticipated temporary failures when it tries to connect to a service or network resource by transparently retrying an operation that has previously failed.

When you use a retry pattern, pay particular attention to issues and considerations. Avoid overwhelming dependent services by implementing the Circuit Breaker pattern. Review and incorporate other best practices for transient fault handling. While calling systems that have throttling pattern implemented, ensure that your retries aren't counter productive.

Samples related to this pattern are available on GitHub.

Uncover retry logic issues

Track the number of transient exceptions and retries over time to uncover issues or failures in your application's retry logic. A trend of increasing exceptions over time might indicate that the service has an issue and might fail. To learn more, see Retry guidance for Azure services.

The Circuit Breaker pattern provides stability while the system recovers from a failure and minimizes the impact on performance. It can help to maintain the response time of the system by quickly rejecting a request for an operation that's likely to fail, rather than waiting for the operation to time out, or never return.

A circuit breaker might be able to test the health of a service by sending a request to an endpoint exposed by the service. The service should return information indicating its status.

This reference implementation on GitHub uses Polly and IHttpClientBuilder to implement the Circuit Breaker pattern.

Configure request timeouts

For a service call or a database call, ensure that appropriate request timeouts are set. Database connection timeouts are typically set to 30 seconds. For guidance on how to troubleshoot, diagnose, and prevent SQL connection errors, see transient errors for SQL Database.

Use design patterns that encapsulate robust timeout strategies like Choreography pattern or Compensating Transaction pattern.

A reference implementation is available on GitHub.

Application health probes

Configure and test health probes for your load balancers and traffic managers. Ensure that your health endpoint checks the critical parts of the system and responds appropriately.

- For Azure Front Door and Azure Traffic Manager, the health probe determines whether to fail over to another region. Your health endpoint should check any critical dependencies that are deployed within the same region.
- For Azure Load Balancer, the health probe determines whether to remove a virtual machine (VM) from rotation. The health endpoint should report the health of the

VM. Don't include other tiers or external services. Otherwise, a failure that occurs outside the VM causes the load balancer to remove the VM from rotation.

Samples related to heath probes are available on GitHub. The samples include:

- ARM template that deploys an Azure Load Balancer and health probes that detect the health of the sample service endpoint.
- An ASP.NET Core Web API that shows configuration of health checks at startup.

Command and query responsibility segregation (CQRS)

Segregate read and write interfaces by implementing the CQRS pattern to achieve the levels of scale and performance needed for your solution.

Related links

- For information on transient faults, see Troubleshoot transient connection errors.
- For guidance on implementing health monitoring in your application, see Health Endpoint Monitoring pattern.

Go back to the main article: Testing for reliability

Next steps

Chaos engineering

Use chaos engineering to test Azure applications

Article • 05/30/2023

Chaos engineering is a methodology that helps developers attain consistent reliability by hardening services against production failures. Another way to think about chaos engineering is that it's about embracing the inherent chaos in complex systems and, through experimentation, growing confidence in your solution's ability to handle it.

A common way to introduce chaos is to deliberately inject faults that cause system components to fail. The goal is to observe, monitor, respond to, and improve your system's reliability under adverse circumstances. For example, taking dependencies offline (stopping API apps, shutting down VMs, etc.), restricting access (enabling firewall rules, changing connection strings, etc.), or forcing failover (database level, Front Door, etc.), is a good way to validate that the application is able to handle faults gracefully.

It's difficult to simulate the characteristics of a service's behavior at scale outside a production environment. The transient nature of cloud platforms can exacerbate this difficulty. Architecting your service to expect failure is a core approach to creating a modern service. Chaos engineering embraces the uncertainty of the production environment and strives to anticipate rare, unpredictable, and disruptive outcomes, so that you can minimize any potential effect on your customers.

Key points

- Increase service resiliency and ability to react to failures.
- Apply chaos principles continuously.
- Create and organize a central chaos engineering team.
- Follow best practices for chaos testing.

Increase resiliency

Chaos engineering is aimed at increasing your service's resiliency and its ability to react to failures. By conducting experiments in a controlled environment, you can identify issues that are likely to arise during development and deployment. During this process, be vigilant in adopting the following guidelines:

- Be proactive.
- Embrace failure.

- Break the system.
- Identify and address single points of failure early.
- Install guardrails and graceful mitigation.
- Minimize the blast radius.
- Build immunity.

Chaos engineering should be an integral part of development team culture and an ongoing practice, not a short-term tactical effort in response to a single outage.

Development team members are partners in the process. They must be equipped with the resources to triage issues, implement the testability that's required for fault injection, and drive the necessary product changes.

When to apply chaos

Ideally, you should apply chaos principles continuously. There's constant change in the environments in which software and hardware run, so monitoring the changes is key. By constantly applying stress or faults on components, you can help expose issues early, before small problems are compounded by many other factors.

Apply chaos engineering principles when you:

- Deploy new code.
- Add dependencies.
- Observe changes in usage patterns.
- Mitigate problems.

Process

Chaos engineering requires specialized expertise, technology, and practices. As with security and performance teams, the model of a central team supporting the service teams is a common, effective approach.

If you plan to practice the simulated handling of potentially catastrophic scenarios under controlled conditions, here's a simplified way to organize your teams:

Attacker	Defender for Cloud
Inject faults	Assess
Provide hints	Analyze
	Mitigate

Goals

- Familiarize team members with monitoring tools.
- Recognize outage patterns.
- Learn how to assess the impact.
- Determine the root cause and mitigate accordingly.
- Practice log analysis.

Overall method

1. Start with a hypothesis.
2. Measure baseline behavior.
3. Inject a fault or faults.
4. Monitor the resulting behavior.
5. Document the process and observations.
6. Identify and act on the result.

Periodically validate your process, architecture choices, and code. By conducting fault-injection experiments, you can confirm that monitoring is in place and alerts are set up, the *directly responsible individual* (DRI) process is effective, and your documentation and investigation processes are up to date. Keep in mind a few key considerations:

- Challenge system assumptions.
- Validate change (topology, platform, resources).
- Use service-level agreement (SLA) buffers.
- Use live-site outages as opportunities.

Best practices

Shift left

Shift-left testing means experiment early, experiment often. Incorporate fault-injection configurations and create resiliency-validation gates during the development stages and in the deployment pipeline.

Shift right

Shift-right testing means that you verify that the service is resilient where it counts in a preproduction or production environment with actual customer load. Adopt a proactive approach as opposed to reacting to failures. Be a part of determining and controlling requirements for the blast radius.

Blast radius

Stop the experiment when it goes beyond scope. Unknown results are an expected outcome of chaos experiments. Strive to achieve balance between collecting substantial result data and affecting as few production users as possible. For an example of this principle in practice, see the Bulkhead pattern article.

Error budget testing

Establish an error budget as an investment in chaos and fault injection. Your error budget is the difference between achieving 100% of the service-level objective (SLO) and achieving the *agreed-upon* SLO.

Considerations for chaos testing

The following questions and answers discuss considerations about chaos engineering, based on its application inside Azure.

- **Have you identified faults that are relevant to the development team?**

  Work closely with development teams to ensure the relevance of the injected failures. Use past incidents or issues as a guide. Examine dependencies, and evaluate the results when those dependencies are removed.

  An external team can't hypothesize faults for your team. A study of failures from an artificial source might be relevant to your team's purposes, but the effort must be justified.

- **Have you injected faults in a way that accurately reflects production failures?**

  Simulate production failures. Treat injected faults in the same way that you treat production-level faults. Enforcing a tighter limit on the blast radius lets you simulate a production environment. Each fault-injection effort must be accompanied by tooling that's designed to inject the types of faults that are relevant to your team's scenarios. Here are two basic ways:
  - Inject faults in a non-production environment, such as Canary or Testing In Production (TIP).
  - Partition the production service or environment.

If the state seems severe, halt all faults and roll back the state to its last-known good configuration.

- **Have you built confidence incrementally?**

  Start by hardening the core, and then expand out in layers. At each point, lock in progress with automated regression tests. Each team should have a long-term strategy based on a progression that makes sense for the team's circumstances.

  By applying the shift left strategy, you can help ensure that any obstacles to developer usage are removed early and the testing results are actionable.

  The process must be very *low tax*. That is, the process must make it easy for developers to understand what happened and to fix the issues. The effort must fit easily into your developers' normal workflow, not burden them with one-off special activities.

Related links

- For information on release testing, see Testing your application and Azure environment.
- Learn more about the Bulkhead pattern.

Go back to the main article: Testing for reliability

Next steps

Testing best practices

Best practices for testing reliability in Azure applications

Article • 05/30/2023

This article lists Azure best practices to enhance the testing of Azure applications for reliability. These best practices are derived from our experience with Azure reliability and the experiences of customers like yourself.

While you design the architecture, focus on implementing practices that meet your business requirements, and ensure that applications run in a healthy state without significant downtime.

Test regularly

Test regularly to validate existing thresholds, targets, and assumptions. Regular testing should be performed as part of each major change and, if possible, on a regular basis. While most testing should be performed within the testing and staging environments, it's often beneficial to also run a subset of tests against the production system.

Test for resiliency

To test resiliency, you should verify how the end-to-end workload performs under intermittent failure conditions. Consider performing the following tests:

- Performance testing
- Simulation testing
- Fault injection testing
- Load testing
- Operational readiness testing
- Failover and failback testing

Design a backup strategy

Design a backup strategy that's tailored for the specific business requirements and circumstances of the application. At a high level, the approaches can be divided into these categories: 1) Redeploy on disaster, 2) Warm spare (Active/Passive), and 3) Hot spare (Active/Active).

Design a disaster recovery strategy

When parts of the Azure network are inaccessible, you might not be able to access your application or data. For this scenario, design a disaster recovery strategy to run most applications with reduced functionality.

Codify steps to failover and fallback

Codify steps, preferably automatically, to failover and fallback the application to the primary region once a failover-triggering issue has been addressed. Doing this should ensure capabilities exist to effectively respond to an outage in a way that limits its impact.

Plan for regional failures

Use Azure to create applications that are distributed across regions. Such distribution helps to minimize the possibility that a failure in one region affects other regions.

Implement retry logic

Track the number of transient exceptions and retries over time to uncover issues or failures in your application's retry logic. A trend of increasing exceptions over time might indicate that the service has an issue and might fail.

Configure and test health probes

Configure and test health probes for your load balancers and traffic managers. Ensure that your health endpoint checks the critical parts of the system and responds appropriately.

Segregate read and write interfaces

By implementing the Command and Query Responsibility Segregation (CQRS) pattern to segregate read and write interfaces, you can achieve levels of scale and performance needed for your solution.

Next steps

Monitoring for reliability

Go back to the main article: Checklist for reliability testing

Monitoring for reliability

Article • 05/30/2023

Monitoring and diagnostics are crucial for reliability. If something fails, you need to know *that* it failed, *when* it failed, and *why*.

Checklist

How do you monitor and measure application health?

- ✓ The application is instrumented with semantic logs and metrics.
- ✓ Application logs are correlated across components.
- ✓ All components are monitored and correlated with application telemetry.
- ✓ Key metrics, thresholds, and indicators are defined and captured.
- ✓ A health model has been defined based on performance, availability, and recovery targets.
- ✓ Azure Service Health events are used to alert on applicable service level events.
- ✓ Azure Resource Health events are used to alert on resource health events.
- ✓ Monitor long-running workflows for failures.

Azure services for monitoring

- Azure Monitor
- Application Insights
- Azure Service Health
- Azure Resource Health
- Azure Resource Manager
- Azure Policy

Reference architecture

- Hybrid availability and performance monitoring
- Unified logging for microservices applications

Related links

- Azure Monitor
- Cloud monitoring guide

Next steps

Monitoring application health for reliability

Monitoring application health for reliability

Article • 05/30/2023

Monitoring and diagnostics are crucial for availability and resiliency. If something fails, you need to know *that* it failed, *when* it failed, and *why*.

Monitoring isn't the same as *failure detection*. For example, your application might detect a transient error and retry, avoiding downtime. It should also log the retry operation so that you can monitor the error rate to get an overall picture of application health.

Key points

- Define alerts that are actionable and effectively prioritized.
- Create alerts that poll for services nearing their limits and quotas.
- Use application instrumentation to detect and resolve performance anomalies.
- Track the progress of long-running processes.
- Troubleshoot issues to gain an overall view of application health.

Alerting

Alerts are notifications of system health issues that are found during monitoring. Alerts only deliver value if they're actionable and effectively prioritized by on-call engineers through defined operational procedures. Present data in a dashboard or email alert format that makes it easy for an operator to notice problems or trends quickly.

Service-level alerts

Use Azure Service Health to respond to *service level* events. Azure Service Health provides a view into the health of Azure services and regions. It issues communications that affect the following services:

- Outages
- Planned maintenance activities
- Other health advisories

Azure Service Health alerts should be configured to operationalize Service Health events. However, Service Health alerts shouldn't be used to detect issues because of associated latencies. There's a five-minute service-level objective (SLO) for automated issues. Many issues require manual interpretation to define a root cause analysis (RCA). Instead, alerts should be used to provide useful information to help interpret issues that have been detected and surfaced through the health model to inform an operational response.

For more information, see Azure Service Health.

Resource-level alerts

Use Azure Resource Health to respond to *resource level* events. Azure Resource Health provides information about the health of individual resources such as a specific virtual machine. It's useful to diagnose unavailable resources.

Azure Resource Health alerts should be configured for specific resource groups and resource types. These alerts should be adjusted to maximize signal to noise ratios. For example, only distribute a notification when a resource becomes unhealthy according to the application health model or due to an Azure platform initiated event.

It's important to consider transient issues when setting an appropriate threshold for resource unavailability. For example, configure an alert for a virtual machine with a threshold of one minute for unavailability before an alert is triggered.

For more information, see Azure Resource Health.

Dashboards

You can also get a full-stack view of application state by using Azure dashboards to create a combined view of monitoring graphs from the following sources:

- Application Insights
- Azure Monitor Logs
- Azure Monitor metrics
- Service Health

Samples

Here are some samples about creating and querying alerts:

- HealthAlerts. A sample about creating resource-level health activity log alerts. The sample uses Azure Resource Manager to create alerts.

- GraphAlertsPsSample. Azure PowerShell and Azure commands that query for alerts generated against your subscription.

Azure subscription and service limits

Azure subscriptions have limits on certain resource types, such as number of resource groups, cores, and storage accounts. To ensure that your application doesn't run up against Azure subscription limits, create alerts that poll for services nearing their limits and quotas.

Address the following subscription limits with alerts.

Individual services

Individual Azure services have consumption limits on:

- Storage
- Throughput
- Number of connections
- Requests per second

Your application fails if it attempts to use resources beyond these limits. This situation results in service throttling and possible downtime.

Depending on the specific service and your application requirements, you can often stay under these limits by *scaling up* or *scaling out*. Scaling up could involve choosing another pricing tier. Scaling out involves adding new instances.

Azure storage scalability and performance targets

Azure allows a maximum number of storage accounts per subscription. If your application requires more storage accounts than are currently available in your subscription, create a new subscription with extra storage accounts. For more information, see Azure subscription and service limits, quotas, and constraints.

Scalability targets for virtual machine disks

An Azure infrastructure as a service (IaaS) virtual machine supports attaching many data disks, depending on several factors. The factors include the virtual machine size and the type of storage account. If your application exceeds the scalability targets for virtual machine disks, provision more storage accounts and create the virtual machine disks there. For more information, see Scalability and performance targets for VM disks.

Virtual machine size

If the actual CPU, memory, disk, and I/O of your virtual machines approach the limits of the virtual machine size, your application might experience capacity issues. To correct the issues, increase the virtual machine size.

If your workload fluctuates over time, consider using Azure Virtual Machine Scale Sets to automatically scale the number of virtual instances. Otherwise, you need to manually increase or decrease the number of virtual machines.

Azure SQL Database

If your Azure SQL Database tier isn't adequate to handle your application's Database Transaction Unit (DTU) requirements, your data use is throttled. For more information on selecting the correct service plan, see Compare vCore and DTU-based purchasing models.

Instrumentation

Use instrumentation to measure the customer experience. Effective instrumentation is vital for detecting and resolving performance anomalies that can affect customer experience and application availability. To build a robust application health model, it's vital that you achieve visibility into the operational state of critical internal dependencies, such as a shared Network Virtual Appliance (NVA) or Azure ExpressRoute connection.

Automated failover and failback systems depend on the correct functioning of monitoring and instrumentation. Dashboards that visualize system health and operator alerts also depend on having accurate monitoring and instrumentation. These elements might fail, miss critical information, or report inaccurate data. If so, an operator might not realize that the system is unhealthy or failing. Make sure that you include monitoring systems in your test plan.

Use applications to track calls to dependent services. Dependency tracking and measuring the duration or status of dependency calls is also vital to measuring overall application health. It should be used to inform a health model for the application.

We recommend collecting and storing logs and key metrics of critical components.

Provide rich instrumentation:

- For failures that are likely, but haven't yet occurred, provide enough data to determine the cause, mitigate the situation, and ensure that the system remains available.
- For failures that have already occurred, the application should return an appropriate error message to the user. The application should attempt to continue running despite reduced functionality.

Monitoring systems should capture comprehensive details so that applications can be restored efficiently. Designers and developers can modify the system to prevent the situation from recurring.

Long-running workflow failures

Long-running workflows often include multiple steps, each of which should be independent.

Track the progress of long-running processes to minimize the likelihood that the entire workflow needs to be rolled back or that multiple compensating transactions need to be run.

> **Tip**
>
> Monitor and manage the progress of long-running workflows by implementing a pattern such as **Scheduler Agent Supervisor**.

Analysis and diagnosis

Analyze data combined in the data stores to troubleshoot issues and gain an overall view of application health. Generally, you can search for and analyze the data in Application Insights and Azure Monitor Logs using Kusto queries. View preconfigured graphs using management solutions. Use Azure Advisor to view recommendations with a focus on resiliency and performance.

Related links

- For information on dashboards, see Azure dashboards.
- For information on virtual machine sizes, see Sizes for virtual machines in Azure.
- For information on scale sets, see Virtual Machine Scale Sets overview.

Next steps

Health modeling for reliability

Go back to the main article: Monitoring for reliability

Health modeling for reliability

Article • 05/30/2023

The health model should be able to surface the health of critical system flows or key subsystems to ensure that appropriate operational prioritization is applied. For example, the health model should be able to represent the current state of the user sign-in transaction flow.

The health model shouldn't treat all failures the same. The health model should distinguish between transient and nontransient faults. It should clearly distinguish between expected-transient but recoverable failures and a true disaster state.

Key points

- Know how to tell if an application is healthy or unhealthy.
- Understand the effects of logs in diagnostic data.
- Ensure the consistent use of diagnostic settings across the application.
- Use critical system flows in your health model.

Healthy and unhealthy states

A health model defines what *healthy* and *unhealthy* states represent for the application. A holistic application health model should be used to quantify what healthy and unhealthy states represent across all application components.

We highly recommend that a *traffic light* model is used to indicate a green or healthy state when key nonfunctional requirements and targets are fully satisfied. Indicate a healthy state when resources are optimally utilized. For example, 95 percent of requests are processed in `<= 500 ms` with AKS node utilization at `x%`. Once established, this health model should inform critical monitoring metrics across system components and operational subsystem composition.

Quantify application states

Application level events should be automatically correlated with resource level metrics to quantify the current application state. The overall health state can be affected by both application level issues and resource level failures.

Application logs

Application logs are an important source of diagnostics data. To gain insight when you need it most, follow these best practices for application logging:

- Use semantic, or *structured*, logging.

  With structured logs, it's easier to automate the consumption and analysis of the log data, which is especially important at cloud scale. We recommend storing Azure resources metrics and diagnostics data in an Azure Monitor Logs workspace rather than in a storage account. This way, you can use Kusto queries to obtain the data you want quickly and in a structured format. You can also use Azure Monitor APIs.

- Log data in the production environment.

  Capture robust data while the application is running in the production environment. You need sufficient information to diagnose the cause of issues in the production state.

- Log events at service boundaries.

  Include a correlation ID that flows across service boundaries. If a transaction flows through multiple services and one of them fails, the correlation ID helps you track requests across your application and pinpoint why the transaction failed.

- Use asynchronous logging.

  Synchronous logging operations sometimes block your application code. This situation causes requests to back up as logs are written. Use asynchronous logging to preserve availability during application logging.

- Separate application logging from auditing.

  Audit records are commonly maintained for compliance or regulatory requirements and must be complete. To avoid dropped transactions, maintain audit logs separately from diagnostic logs.

All application resources should be configured to route diagnostic logs and metrics to the chosen log aggregation technology. Azure Policy should also be used to ensure the consistent use of diagnostic settings across the application, which enforces the desired configuration for each Azure service.

Telemetry correlation should be used to ensure that transactions can be mapped through the end-to-end application and critical system flows. This process is vital to root cause analysis (RCA) for failures. Platform level metrics and logs, such as CPU percentage, network in/out, and disk operations/sec, should be collected from the application to inform a health model and to detect and predict issues. This approach can also help to distinguish between transient and nontransient faults.

White-box and black-box monitoring

Use white-box monitoring to instrument the application with semantic logs and metrics. Application level metrics and logs, such as current memory consumption or request latency, should be collected from the application to inform a health model and to detect and predict issues.

Use black-box monitoring to measure platform services and the resulting customer experience. Black-box monitoring tests externally visible application behavior without knowledge of the internals of the system. This approach is common for measuring customer-centric service-level indicators (SLIs), service-level objectives (SLOs), and service-level agreements (SLAs).

Use critical system flows in the health model

The health model should be able to surface the respective health of critical system flows or key subsystems to ensure that appropriate operational prioritization is applied. For example, the health model should be able to represent the current state of the user sign-in transaction flow.

Create good health probes

The health and performance of an application can degrade over time. That degradation might not be noticeable until the application fails.

Implement probes or check functions. Run them regularly from outside the application. These checks can be as simple as measuring response time for the application as a whole, for individual parts of the application, for specific services that the application uses, or for separate components.

Check functions can run processes to ensure that they produce valid results, measure latency and check availability, and extract information from the system.

The HealthProbesSample sample shows how to set up health probes. It provides an Azure Resource Manager template to set up the infrastructure. A load balancer accepts public requests and load balances to a set of virtual machines. The health probe is set up so that it can check for service's path */Health*.

Related links

- For information on monitoring metrics, see Azure Monitor Metrics overview.
- For information on using Application Insights, see Application Insights.

Next steps

Monitoring best practices for reliability

Go back to the main article: Monitoring for reliability

Best practices for monitoring reliability in Azure applications

Article • 05/30/2023

This article describes Azure best practices to enhance monitoring Azure applications for reliability. These best practices are derived from our experience with Azure reliability and the experiences of customers like yourself.

Implement these best practices for monitoring and alerts in your application so that you can detect failures and alert an operator to fix them.

Implement health probes and check functions

Run health probes and check functions regularly from outside the application to identify degradation of application health and performance.

Check long-running workflows

Catching issues early can minimize the need to roll back the entire workflow or run multiple compensating transactions.

Maintain application logs

- Log applications in production and at service boundaries.
- Use semantic and asynchronous logging.
- Separate application logs from audit logs.

Measure remote call statistics

Measure remote call statistics, and share the data with the application team. This data gives the team an instantaneous view into application health and summarizes remote call metrics, such as latency, throughput, and errors in the 99 and 95 percentiles. Perform statistical analysis on the metrics to uncover errors that occur in each percentile.

Track transient exceptions and retries

A trend of increasing exceptions over time indicates that the service has an issue and might fail. Track transient exceptions and retries over an appropriate time frame to prevent failure.

Set up an early warning system

Identify the key performance indicators (KPIs) of an application's health, such as transient exceptions and remote call latency. Set appropriate threshold values for each KPI. Send an alert to operations when the threshold value is reached.

Operate within Azure subscription limits

Azure subscriptions have limits on certain resource types, such as the number of resource groups, cores, and storage accounts. Watch your use of resource types.

Monitor third-party services

Log your invocations and correlate them with your application's health and diagnostic logging using a unique identifier.

Train multiple operators

Train multiple operators to monitor the application and to perform manual recovery steps. Make sure there's always at least one trained operator active.

Next steps

Go back to the main article: Monitoring for reliability

Reliability patterns

Article • 05/30/2023

Availability

Availability is measured as a percentage of uptime, and defines the proportion of time that a system is functional and working. Availability is affected by system errors, infrastructure problems, malicious attacks, and system load. Cloud applications typically provide users with a service-level agreement (SLA), which means that applications must be designed and implemented to maximize availability.

Pattern	Summary
Deployment Stamps	Deploy multiple independent copies of application components, including data stores.
Geode	Deploy backend services into a set of geographical nodes, each of which can service any client request in any region.
Health Endpoint Monitoring	Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.
Queue-Based Load Leveling	Use a queue that acts as a buffer between a task and a service that it invokes to smooth intermittent heavy loads.
Throttling	Control the consumption of resources by an instance of an application, an individual tenant, or an entire service.

To mitigate against availability risks from malicious distributed denial of service (DDoS) attacks, implement the native Azure DDoS protection service or a third-party capability.

High availability

Azure infrastructure is composed of geographies, regions, and availability zones. These divisions limit the radius of a failure and therefore limit potential effect on customer applications and data. The Azure availability zones construct was developed to provide a software and networking solution to protect against datacenter failures and to provide increased high availability. With high availability architecture, there's a balance between high resilience, low latency, and cost.

Pattern	Summary

Pattern	Summary
Deployment Stamps	Deploy multiple independent copies of application components, including data stores.
Geode	Deploy backend services into a set of geographical nodes. Each node can service any client request in any region.
Health Endpoint Monitoring	Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.
Bulkhead	Isolate elements of an application into pools. If one element fails, the others continue to function.
Circuit Breaker	Handle faults that might take a variable amount of time to fix when connecting to a remote service or resource.

Resiliency

Resiliency is the ability of a system to gracefully handle and recover from failures, both inadvertent and malicious.

In cloud hosting, applications are often multi-tenant, use shared platform services, compete for resources and bandwidth, communicate over the Internet, and run on commodity hardware. This situation means there's an increased likelihood for both transient and permanent faults to arise. The connected nature of the internet and the rise in sophistication and volume of attacks increase the likelihood of a security disruption.

To detect failures and recovering quickly and efficiently, it's necessary to maintain resiliency.

Pattern	Summary
Bulkhead	Isolate elements of an application into pools. If one element fails, the others continue to function.
Circuit Breaker	Handle faults that might take a variable amount of time to fix when connecting to a remote service or resource.
Compensating Transaction	Undo the work performed by a series of steps, which together define an eventually consistent operation.
Health Endpoint Monitoring	Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.

Pattern	Summary
Leader Election	Coordinate the actions performed by a collection of collaborating task instances in a distributed application by electing one instance as the leader. The leader assumes responsibility for managing the other instances.
Queue-Based Load Leveling	Use a queue that acts as a buffer between a task and a service that it invokes. This queue smooths intermittent heavy loads.
Retry	Enable an application to handle anticipated, temporary failures when it tries to connect to a service or network resource by transparently retrying an operation that's previously failed.
Scheduler Agent Supervisor	Coordinate a set of actions across a distributed set of services and other remote resources.

Security documentation

Apply security principles to your architecture to protect against attacks on your data and systems.

Key points

QUICKSTART

Overview

Principles

Design governance

Identity checklist

Network checklist

Data checklist

Build and deploy checklist

Monitor checklist

Governance, risk, and compliance

TRAINING

Security

VIDEO

Cloud Security Posture Management (CSPM) with Microsoft Defender for Cloud

Design for security

GET STARTED

Plan segmentation

Consider team roles and responsibilities

Implement network segmentation

Encrypt data

Understand application security

Understand application security

CONCEPT

Support segmentation with management groups

Identity management and access control best practices

Network best practices

Encryption best practices

Analyze application threats

Gather compliance requirements

Secure application configuration and dependencies

TRAINING

Design and implement network security

WHAT'S NEW

What's new in Azure Network Security?

Secure build and deploy

OVERVIEW

Consider secure deployment governance

Understand infrastructure provisioning

Deploy code

CONCEPT

Minimize access

Understand Infrastructure as Code (IaC)

Roll back and roll forward

Scanning credentials

VIDEO

Ask the right questions about secure application development on Azure

Security monitoring

OVERVIEW

Security monitoring tools in Azure

Security operations in Azure

Validate and test security design

Review and audit security posture

Check for identity, network, and data risks

CONCEPT

Prevent, detect, and respond to threats

Security operations best practices

Penetration testing (pentesting)

Review critical access

Review identity risks

VIDEO

Improving app security with Application Security Groups

Optimize security

CONCEPT

Prioritize security best practices investments

Use Azure Secure Score

Manage connected tenants

Designate clear lines of responsibility

VIDEO

Enabling secure remote work with using Windows Virtual Desktop

Security tools and services

REFERENCE

Azure AD Conditional Access

Azure AD Connect

Azure AD Privileged Identity Management (PIM)

Azure Key Vault

Azure Resource Manager

Azure role-based access control (RBAC)

Azure Storage Service Encryption

Microsoft Azure Well-Architected Review

Multifactor Authentication

Passwordless

Security APIs

REFERENCE

Microsoft Defender for Cloud

Adaptive Network Hardenings

Alerts

Iot Security Solution

Secure Scores

Overview of the security pillar

Article • 11/30/2022

Information security has always been a complex subject, and it evolves quickly with the creative ideas and implementations of attackers and security researchers. The origin of security vulnerabilities started with identifying and exploiting common programming errors and unexpected edge cases. However over time, the attack surface that an attacker may explore and exploit has expanded well beyond these common errors and edge cases. Attackers now freely exploit vulnerabilities in system configurations, operational practices, and the social habits of the systems' users. As system complexity, connectedness, and the variety of users increase, attackers have more opportunities to identify unprotected edge cases. Attackers can *hack* systems into doing things they weren't designed to do.

Security is one of the most important aspects of any architecture. It provides the following assurances against deliberate attacks and abuse of your valuable data and systems:

- Confidentiality
- Integrity
- Availability

Losing these assurances can negatively affect your business operations and revenue, and your organization's reputation. For the security pillar, we'll discuss key architectural considerations and principles for security and how they apply to Azure.

The security of complex systems depends on understanding the business context, social context, and technical context. As you design your system, cover these areas:

Understanding an IT solution as it interacts with its surrounding environment holds the key to preventing unauthorized activity and to identifying anomalous behavior that may represent a security risk.

Another key factor in success: Adopt a mindset of assuming failure of security controls. Assuming failure allows you to design compensating controls that limit risk and damage if a primary control fails.

Assuming failures can be referred to as *assume breach* or *assume compromise*. Assume breach is closely related to the *Zero Trust* approach of continuously validating security assurances. The Zero Trust approach is described in the Security Design Principles section in more detail.

Cloud architectures can help simplify the complex task of securing an enterprise estate through specialization and shared responsibilities:

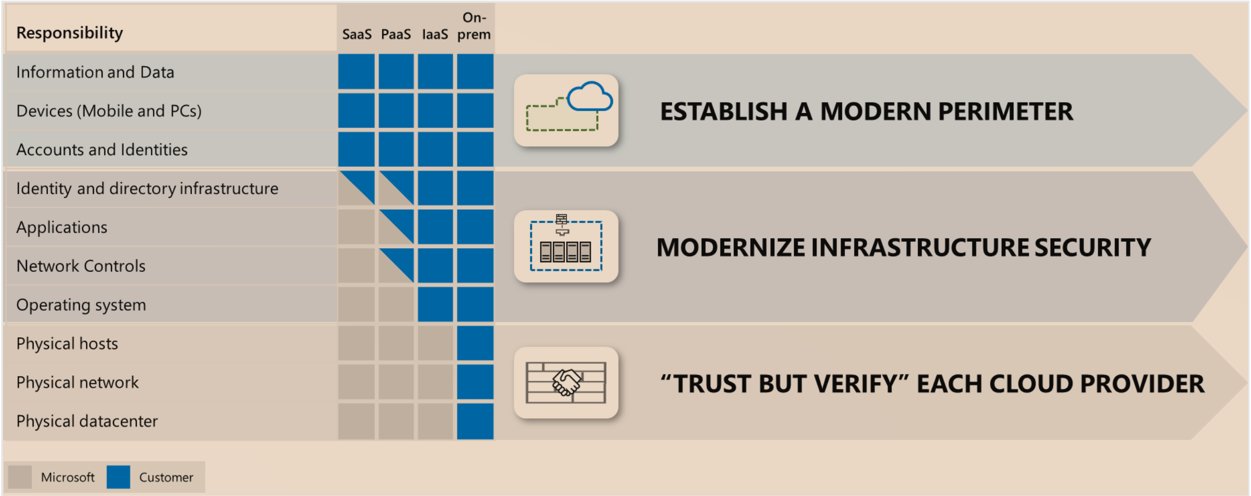
Specialization: Specialist teams at cloud providers can develop advanced capabilities to operate and secure systems on behalf of organizations. This approach is preferable to numerous organizations individually developing deep expertise on managing and securing common elements, such as:

- Datacenter physical security
- Firmware patching
- Hypervisor configuration

The economies of scale allow cloud provider specialist teams to invest in optimization of management and security that far exceeds the ability of most organizations.

Cloud providers must be compliant with the same IT regulatory requirements as the aggregate of all their customers. Providers must develop expertise to defend against the aggregate set of adversaries who attack their customers. As a consequence, the default security posture of applications deployed to the cloud is frequently much better than that of applications hosted on-premises.

Shared Responsibility Model: As computing environments move from customer-controlled datacenters to the cloud, the responsibility of security also shifts. Security of the operational environment is now a concern shared by both cloud providers and customers. Organizations can reduce focus on activities that aren't core business competencies by shifting these responsibilities to a cloud service like Azure. Depending on the specific technology choices, some security protections will be built into the particular service, while addressing others will remain the customer's responsibility. To ensure that proper security controls are provided, organizations must carefully evaluate the services and technology choices.



Shared Responsibility and Key Strategies:

After reading this document, you'll be equipped with key insights about how to improve the security posture of your architecture.

As part of your architecture design, you should consider all relevant areas that affect the success of your application. While this article is concerned primarily with security principles, you should also prioritize other requirements of a well-designed system, such as:

- Availability
- Scalability
- Costs
- Operational characteristics (trading off one over the other as necessary)

Consistently sacrificing security for gains in other areas isn't advisable because security risks tend to increase dynamically over time.

Increasing security risks result in three key strategies:

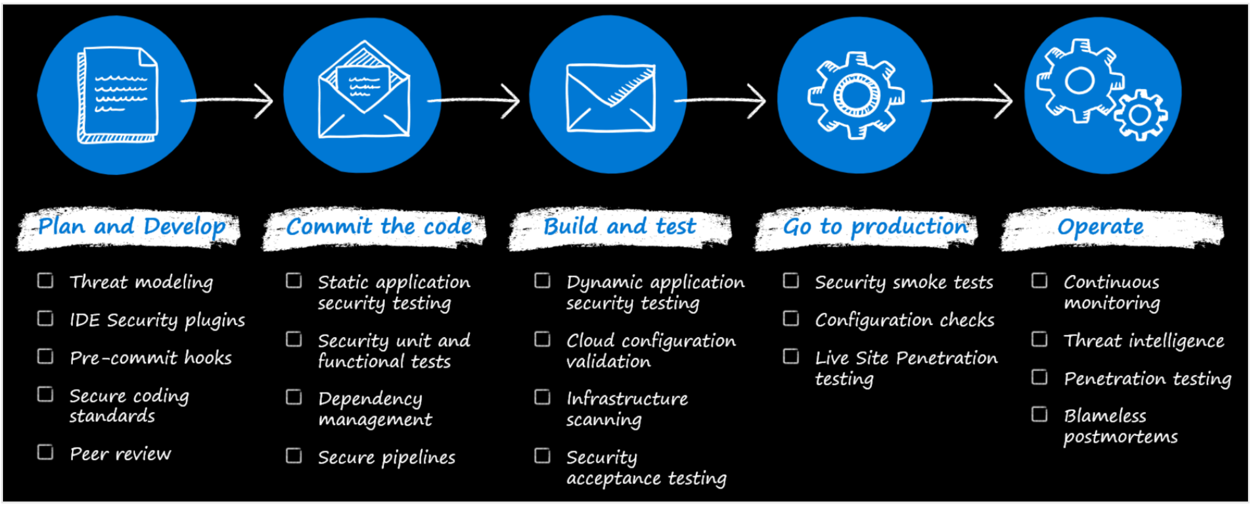
- **Establish a modern perimeter:** For the elements that your organization controls to ensure you have a consistent set of controls (a perimeter) between those assets and the threats to them. Perimeters should be designed based on intercepting authentication requests for the resources (identity controls) versus intercepting network traffic on enterprise networks. This traditional approach isn't feasible for enterprise assets outside the network.

More on perimeters and how they relate to Zero Trust and Enterprise Segmentation are in the Governance, Risk, and Compliance and Network Security & Containment sections.

- **Modernize infrastructure security:** For operating systems and middleware elements that legacy applications require, take advantage of cloud technology to reduce security risk to the organization. For example, knowing whether all servers in a physical datacenter are updated with security patches has always been challenging because of discoverability. Software-defined datacenters allow easy and rapid discovery of all resources. This rapid discovery enables technology like Microsoft Defender for Cloud to measure quickly and accurately the patch state of all servers and remediate them.

- **"Trust but verify" each cloud provider:** For the elements, which are under the control of the cloud provider. You should ensure the security practices and regulatory compliance of each cloud provider (large and small) meet your requirements.

To assess your workload using the tenets found in the Microsoft Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.



We cover the following areas in the security pillar of the Microsoft Azure Well-Architected Framework:

Security Topic	Description
Security design principles	These principles describe a securely architected system hosted on cloud or on-premises datacenters, or a combination of both.
Governance, risk, and compliance	How is the organization's security going to be monitored, audited, and reported? What types of risks does the organization face while trying to protect identifiable information, Intellectual Property (IP), financial information? Is there specific industry, government, or regulatory requirements that dictate or provide recommendations on criteria that your organization's security controls must meet?
Regulatory compliance	Governments and other organizations frequently publish standards to help define good security practices (due diligence) so that organizations can avoid being negligent in security.

Security Topic	Description
Administration	Administration is the practice of monitoring, maintaining, and operating Information Technology (IT) systems to meet service levels that the business requires. Administration introduces some of the highest impact security risks because performing these tasks requires privileged access to a broad set of these systems and applications.
Applications and services	Applications and the data associated with them ultimately act as the primary store of business value on a cloud platform.
Identity and access management	Identity provides the basis of a large percentage of security assurances.
Information protection and storage	Protecting data at rest is required to maintain confidentiality, integrity, and availability assurances across all workloads.
Network security and containment	Network security has been the traditional linchpin of enterprise security efforts. However, cloud computing has increased the requirement for network perimeters to be more porous and many attackers have mastered the art of attacks on identity system elements (which nearly always bypass network controls).
Security Operations	Security operations maintain and restores the security assurances of the system as live adversaries attack it. The tasks of security operations are described well by the NIST Cybersecurity Framework functions of Detect, Respond, and Recover.

Identity management

Consider using Azure Active Directory (Azure AD) to authenticate and authorize users. Azure AD is a fully managed identity and access management service. You can use it to create domains that exist purely on Azure, or integrate with your on-premises Active Directory identities.

Azure AD is also used by:

- Microsoft 365
- Dynamics 365
- Many third-party applications

For consumer-facing applications, Azure Active Directory B2C lets users authenticate with their existing social accounts, such as:

- Facebook

- Google
- LinkedIn

Users can also create a new user account managed by Azure AD.

If you want to integrate an on-premises Active Directory environment with an Azure network, several approaches are possible, depending on your requirements. For more information, reference Identity Management reference architectures.

Protect your infrastructure

Control access to the Azure resources that you deploy. Every Azure subscription has a trust relationship with an Azure AD tenant.

Use Azure role-based access control (Azure RBAC role) to grant users within your organization the correct permissions to Azure resources. Grant access by assigning Azure roles to users or groups at a certain scope. The scope can be a:

- Subscription
- Resource group
- Single resource

Audit all changes to infrastructure.

Application security

In general, the security best practices for application development still apply in the cloud. Best practices include:

- Encrypt data in-transit with the latest supported `TLS` versions
- Protect against `CSRF` and `XSS` attacks
- Prevent SQL injection attacks

Cloud applications often use managed services that have access keys. Never check these keys into source control. Consider storing application secrets in Azure Key Vault.

Data sovereignty and encryption

Make sure that your data remains in the correct geopolitical zone when using Azure data services. Azure's geo-replicated storage uses the concept of a paired region in the same geopolitical region.

Use Key Vault to safeguard cryptographic keys and secrets. By using Key Vault, you can encrypt keys and secrets by using keys that are protected by hardware security modules (HSMs). Many Azure storage and DB services support data encryption at rest, including:

- Azure Storage
- Azure SQL Database
- Azure Synapse Analytics
- Azure Cosmos DB

Security resources

- Microsoft Defender for Cloud provides integrated security monitoring and policy management for your workload.
- Azure Security Documentation
- Microsoft Trust Center

The security pillar is part of a comprehensive set of security guidance that also includes:

- Security in the Microsoft Cloud Adoption Framework for Azure: A high-level overview of a cloud security end state.
- Security architecture design: Implementation-level journey of our security architectures.
  - Browse our security architectures
- Azure security benchmarks: Prescriptive best practices and controls for Azure security.
- End-to-end security in Azure: Documentation that introduces you to the security services in Azure.
- Top 10 security best practices for Azure: Top Azure security best practices that Microsoft recommends based on lessons learned across customers and our own environments.
- Microsoft Cybersecurity Architectures: The diagrams describe how Microsoft security capabilities integrate with Microsoft platforms and 3rd-party platforms.

Next step

Principles

Security design principles

Article • 11/30/2022

Security design principles describe a securely architected system hosted on cloud or on-premises datacenters (or a combination of both). Application of these principles dramatically increases the likelihood your security architecture assures confidentiality, integrity, and availability.

To assess your workload using the tenets found in the Azure Well-Architected Framework, reference the Microsoft Azure Well-Architected Review.

The following design principles provide:

- Context for questions
- Why a certain aspect is important
- How an aspect is applicable to Security

These critical design principles are used as lenses to assess the Security of an application deployed on Azure. These lenses provide a framework for the application assessment questions.

Plan resources and how to harden them

Recommendations:

- Consider security when planning workload resources.
- Understand how individual cloud services are protected.
- Use a service enablement framework to evaluate.

Automate and use least privilege

Recommendations:

- Implement least privilege throughout the application and control plane to protect against data exfiltration and malicious actor scenarios.
- Drive automation through DevSecOps to minimize the need for human interaction.

Classify and encrypt data

Recommendations:

- Classify data according to risk.
- Apply industry-standard encryption at rest and in transit, which ensures keys and certificates are stored securely and managed properly.

Monitor system security, plan incident response

Recommendations:

- Correlate security and audit events to model application health.
- Correlate security and audit events to identify active threats.
- Establish automated and manual procedures to respond to incidents.
- Use security information and event management (SIEM) tooling for tracking.

Identify and protect endpoints

Recommendations:

- Monitor and protect the network integrity of internal and external endpoints through security appliances or Azure services, such as:
  - Firewalls
  - Web application firewalls
- Use industry standard approaches to protect against common attack vectors, such as distributed denial of service (DDoS) attacks like SlowLoris.

Protect against code-level vulnerabilities

Recommendations:

- Identify and mitigate code-level vulnerabilities, such as cross-site scripting and structured query language (SQL) injection.
- In the operational lifecycle, regularly incorporate:
  - Security fixes
  - Codebase and dependency patching

Model and test against potential threats

Recommendations:

- Establish procedures to identify and mitigate known threats.
- Use penetration testing to verify threat mitigation.
- Use static code analysis to detect and prevent future vulnerabilities.

- Use code scanning to detect and prevent future vulnerabilities

Next step

Design governance

Governance, risk, and compliance

Article • 04/19/2023

As part of overall design, prioritize where to invest the available resources: finances, people, and time. Constraints on resources also affect the security implementation across the organization. To achieve an appropriate return on investment (ROI) on security, the organization needs to first understand and define its security priorities.

- **Governance:** How does the organization monitor, audit, and report on its security? The design and implementation of security controls within an organization are only the beginning of the story. How does the organization know that things are actually working? Are they improving? Are there new requirements? Is there mandatory reporting? Similar to compliance, there might be external industry, government, or regulatory standards that need to be considered.

- **Risk:** What risk types does the organization face while trying to protect identifiable information, intellectual property (IP), and financial information? Who might be interested or could use this information if it's stolen, such as external and internal threats and unintentional or malicious threats? A commonly forgotten but important consideration within risk is addressing disaster recovery and business continuity.

- **Compliance:** Is there a specific industry, government, or regulatory requirements that dictate or provide recommendations on criteria that your organization's security controls must meet? Examples of such standards, organizations, controls, and legislation include ISO27001, NIST, and PCI-DSS.

The collective role of organization(s) is to manage the security standards of the organization through their lifecycle:

- **Define:** Set organizational policies for operations, technologies, and configurations based on internal factors (business requirements, risks, and asset evaluation) and external factors (benchmarks, regulatory standards, and threat environment).

- **Improve:** Continually push security standards incrementally towards the ideal state to ensure continual risk reduction.

- **Sustain:** Ensure the security posture doesn't degrade naturally over time by instituting auditing and monitoring compliance with organizational standards.

Take action

Establish security best practices across all workloads

Establish governance best practices to detect, enforce, and automate governance decisions

Prioritize security best practices investments

Security best practices are ideally applied proactively and completely to all systems as you build your cloud program, but this ideal isn't the reality for most enterprise organizations. Business goals, project constraints, and other factors often cause organizations to balance security risk against other risks and apply a subset of best practices at any given point.

We recommend applying as many of the best practices as early as possible, and then working to retrofit any gaps over time as you mature your security program. Your security program should include review, prioritization, and proactive application of best practices to cloud resources. We recommend evaluating the following considerations when prioritizing which to follow first:

- **High business impact and highly exposed systems:** These include systems with direct intrinsic value and the systems that provide attackers a path to them. For more information, see Identify and classify business critical applications.
- **Easiest to implement mitigations:** Identify quick wins by prioritizing the best practices, which your organization can execute quickly because you already have the required skills, tools, and knowledge to do it. For example, implementing a web app firewall (WAF) to protect a legacy application. Be careful not to exclusively use (or overuse) this short-term prioritization method. Doing so can increase your risk by preventing your program from growing and leaving critical risks exposed for extended periods.

Use the provided lists of prioritized security initiatives to help organizations start with these decisions. The provided lists are based on Microsoft's experience with threats and mitigation initiatives in internal environments and across Microsoft's customers. See Module 4a of the Microsoft CISO Workshop.

Take action

Security in the Microsoft Cloud Adoption Framework for Azure

What is an Azure landing zone?

Checklist

What considerations for compliance and governance did you make?

- ✓ Create a landing zone for the workload. The infrastructure must have appropriate controls and be repeatable with every deployment.
- ✓ Enforce creation and deletion of services and their configuration through Azure Policy.
- ✓ Ensure consistency across the enterprise by applying policies, permissions, and tags across all subscriptions through careful implementation of root management group.
- ✓ Understand regulatory requirements and operational data that might be used for audits.
- ✓ Continuously monitor and assess the compliance of your workload. Perform regular attestations to avoid fines.
- ✓ Review and apply recommendations from Azure.
- ✓ Remediate basic vulnerabilities to keep the attacker costs high.

In this section

Follow these questions to assess the workload at a deeper level.

Assessment	Description
Are there any regulatory requirements for this workload?	Understand all regulatory requirements. Check the Microsoft Trust Center for the latest information, news, and best practices in security, privacy, and compliance.
Is the organization using a landing zone for this workload?	Consider the security controls placed on the infrastructure into which the workload is deployed.
Do you have a segmentation strategy?	Reference model and strategies of how the functions and teams can be segmented.
Are you using management groups as part of your segmentation strategy?	Strategies using management groups to manage resources across multiple subscriptions consistently and efficiently.
What security controls do you have in place for access to Azure infrastructure?	Guidance on reducing risk exposure in scope and time when configuring critical impact accounts such as Administrators.

Azure security benchmark

The Azure Security Benchmark includes a collection of high-impact security recommendations you can use to help secure the services you use in Azure:

> The questions in this section are aligned to these controls:
>
> - Governance and strategy
> - Posture and vulnerability management

Reference architecture

Here are some reference architectures related to governance:

Cloud Adoption Framework enterprise-scale landing zone architecture

Next steps

Provide security assurance through identity management to authenticate and grant permission to users, partners, customers, applications, services, and other entities.

Identity and access management

Related links

> Go back to the main article: Security

Regulatory compliance

Article • 11/30/2022

A workload can have regulatory requirements, which may mandate that operational data, such as application logs and metrics, remain within a certain geo-political region.

These requirements may need strict security measures that affect the overall architecture, the selection, and configuration of specific PaaS, and SaaS services. The requirements also have implications for how the workload should be operationalized.

Key points

- ✓ Make sure that all regulatory and governance requirements are known, and well understood.
- ✓ Periodically perform external and, or internal workload security audits.
- ✓ Have compliance checks as part of the workload operations.
- ✓ Use Microsoft Trust Center.

Review the requirements

Regulatory organizations frequently publish standards and updates to help define good security practices so that organizations can avoid negligence. The purpose and scope of these standards, and regulations vary. The security requirements, however, can influence the design for data protection and retention, network access, and system security.

Knowing whether your cloud resources are in compliance with standards mandated by governments or industry organizations is essential in today's globalized world.

For example, a workload that handles credit card transactions is subject to the Payment Card Industry (PCI) standard. One of the requirements prohibits access between the internet and any system component in the cardholder data environment.

To provide a restrictive environment, you can choose to do the following:

- Host the workload in different Azure compute options that supports bring your own VNet.
- Remove any internet-facing endpoints by using Private Endpoints.
- Use network security groups (NSGs) rules that define authorized inbound and outbound access.

Noncompliance can lead to fines or other business impact. Work with your regulators and carefully review the standard to understand both the intent and the literal wording of each requirement. Here are some questions that may help you understand each requirement.

- How is compliance measured?
- Who approves that the workload meets the requirements?
- Are there processes for obtaining attestations?
- What are the documentation requirements?

Suggested action

Use Microsoft Defender for Cloud to assess your current compliance score and to identify the gaps.

Learn more

Tutorial: Improve your regulatory compliance

Use the Microsoft Trust Center

Keep checking the Microsoft Trust Center for the latest information, news, and best practices in security, privacy, and compliance.

- **Data governance**. Focus on protecting information in cloud services, mobile devices, workstations, or collaboration platforms. Build the security strategy by classifying and labeling information. Use strong access control and encryption technology.

- **Compliance offerings**. Microsoft offers a comprehensive set of compliance offerings to help your organization follow national, regional, and industry-specific requirements governing the collection and use of data. For information, see Compliance offerings.

- **Compliance score**. Use Microsoft Compliance Score to assess your data protection controls on an ongoing basis. Act on the recommendations to make progress toward compliance.

- **Audit reports**. Use audit reports to stay current on the latest privacy, security, and compliance-related information for Microsoft's cloud services. See Audit Reports.

- **Shared responsibility**. The workload can be hosted on Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS), or in an on-premises

datacenter. Have a clear understanding about the portions of the architecture you're responsible for versus Azure. Whatever the hosting model, the following responsibilities are always retained by you:
   - Data
   - Endpoints
   - Account
   - Access management

   For more information, reference Shared responsibility in the cloud.

Elevated security capabilities

Consider whether to use specialized security capabilities in your enterprise architecture.

Dedicated HSMs and Confidential Computing have the potential to enhance security and meet regulatory requirements, but can introduce complexity that may negatively impact your operations and efficiency.

Suggested actions

We recommend careful consideration and judicious use of these security measures as required:

- **Dedicated Hardware Security Modules (HSMs)**
  Dedicated Hardware Security Modules (HSMs) may help meet regulatory or security requirements.

- **Confidential Computing**
  Confidential Computing may help meet regulatory or security requirements.

Learn more about elevated security capabilities for Azure workloads.

Operational considerations

Regulatory requirements may influence the workload operations. For example, there might be a requirement that operational data, such as application logs and metrics, remain within a certain geo-political region.

Consider automation of deployment and maintenance tasks. Automation reduces security and compliance risk by limiting opportunity to introduce human errors during manual tasks.

Related links

Azure maintains a compliance portfolio that covers US government, industry specific, and region/country standards. For more information, reference Azure compliance offerings.

Monitor the compliance of the workload to check if the security controls are aligned to the regulatory requirements. For more information, reference Security audits.

> Go back to the main article: Governance

Next

Azure landing zone

Azure landing zone integration

Article • 04/19/2023

From a workload perspective, a *landing zone* refers to a prepared platform into which the application gets deployed. A landing zone implementation can have compute, data sources, access controls, and networking components already provisioned. With the required plumbing ready in place; the workload needs to plug into it. When considering the overall security, a landing zone offers centralized security capabilities that adds a threat mitigation layer for the workload. Implementations can vary but here are some common strategies that enhance the security posture.

- Isolation through segmentation. You can isolate assets at several layers from Azure enrollment down to a subscription that has the resources for the workload. This strategy of having resources within a boundary that is separate from other parts of the organization is an effective way of detecting and containing adversary movements.

- Consistent adoption of organizational policies. Policies govern which resources can be used and their usage limits. Policies also provide identity controls. Only authenticated and authorized entities are allowed access. This approach decouples the governance requirements from the workload requirements. It's crucial that a landing zone is handed over to the workload owner with the security guardrails deployed.

- Configurations that align with principles of Zero Trust . For instance an implementation might have network connectivity to on-premises data centers. When designing networking controls, the landing zone may apply the least-privilege principle by opening communication paths only when necessary and only to trusted entities.

The preceding examples are conceptually simple but the implementation can get complicated for an enterprise-scale deployment. Azure landing zone as part of the Cloud Adoption Framework (CAF) provides architecture guidance about identity and access management, networking, and other design areas necessary to achieve an optimal implementation.

Learn more

What is an Azure landing zone?

Landing zone implementation options

Increase automation with Azure Blueprints

Use Azure's native automation capabilities to increase consistency, compliance, and deployment speed for workloads. A recommended way to implement a landing zone is with Azure Blueprints and Azure Policies.

Automation of deployment and maintenance tasks reduces security and compliance risk by limiting opportunity to introduce human errors during manual tasks. This will also allow both IT Operations teams and security teams to shift their focus from repeated manual tasks to higher value tasks like enabling developers and business initiatives, protecting information, and so on.

Utilize the Azure Blueprint service to rapidly and consistently deploy application environments that are compliant with your organization's policies and external regulations. Azure Blueprint Service automates deployment of environments including Azure roles, policies, resources, such as virtual machines, networking, storage, and more. Azure Blueprints builds on Microsoft's significant investment into the Azure Resource Manager to standardize resource deployment in Azure and enable resource deployment and governance based on a desired-state approach. You can use built in configurations in Azure Blueprint, make your own, or just use Resource Manager scripts for smaller scope.

Several Security and Compliance Blueprints samples are available to use as a starting template.

Enforce policy compliance

Organizations of all sizes will have security compliance requirements. Industry, government, and internal corporate security policies all need to be audited and enforced. Policy monitoring is critical to check that initial configurations are correct and that it continues to be compliant over time.

In Azure, you can take advantage of Azure Policy to create and manage policies that enforce compliance. Like Azure Blueprints, Azure Policies are built on the underlying Azure Resource Manager capabilities in the Azure platform (and Azure Policy can also be assigned via Azure Blueprints).

For more information on how to do this in Azure, please review Tutorial: Create and manage policies to enforce compliance.

How do you consistently deploy landing zones that follow organizational policies?

Key Azure services that can help in creating a landing zone:

- Azure Blueprints sketches a solution's design parameters based on an organization's standards, patterns, and requirements.
- Azure Resource Manager template specs stores an Azure Resource Manager template (ARM template) in Azure for later deployment.
- Azure Policy enforces organizational standards and to assess compliance at-scale.
- Azure AD and Azure role-based access control (Azure RBAC) work in conjunction to provide identity and access controls.
- Microsoft Defender for Cloud

Architecture

For information about an enterprise-scale reference architecture, see Cloud Adoption Framework enterprise-scale landing zone architecture. The architecture provides considerations in these critical design areas:

- Enterprise Agreement (EA) enrollment and Azure Active Directory tenants
- Identity and access management
- Management group and subscription organization
- Network topology and connectivity
- Management and monitoring
- Business continuity and disaster recovery
- Security, governance, and compliance
- Platform automation and DevOps

Next

Use management groups to manage resources across multiple subscriptions consistently and efficiently.

Management groups

> Back to the main article: Governance

Segmentation strategies

Article • 04/19/2023

Segmentation refers to the isolation of resources from other parts of the organization. Segmentation is an effective way of detecting and containing adversary movements.

One approach to segmentation is network isolation. This approach isn't ideal, because different technical teams may not be aligned with the business use cases and application workloads. One outcome of such a mismatch is complexity, as especially seen with on-premises networking, and can lead to reduced velocity, or in worse cases, broad network firewall exceptions. Although network control should be considered as a segmentation strategy, it should be part of a unified segmentation strategy.

Network security has been the traditional linchpin of enterprise security efforts. However, cloud computing has increased the requirement for network perimeters to be more porous. Many attackers have mastered the art of attacks on identity system elements (which nearly always bypass network controls). These factors have increased the need to focus primarily on identity-based access controls to protect resources rather than network-based access controls.

An effective segmentation strategy guides *all* technical teams (IT, security, applications) to consistently isolate access using networking, applications, identity, and any other access controls. The strategy should aim to:

- Minimize operational friction by aligning to business practices and applications
- Contain risk by adding cost to attackers. You can contain risk by:
  - Isolating sensitive workloads from compromise by other assets.
  - Isolating high-exposure systems from being used as a pivot to other systems.
- Monitor operations that might lead to potential violation of the integrity of the segments (account usage, unexpected traffic).

Here are some recommendations for creating a unified strategy:

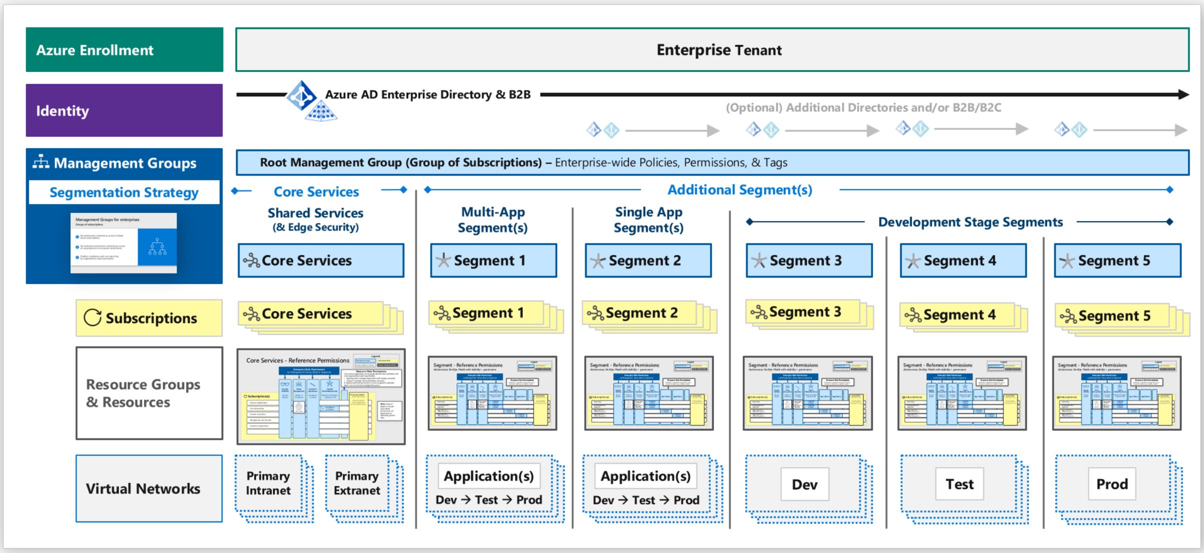
- Ensure alignment of technical teams to a single strategy based on assessing business risks.
- Establish a modern perimeter based on zero-trust principles, focused on identity, devices, applications, and other signals. Establishing modern perimeters helps to overcome limitations of network controls in protecting from new resources and attack types.
- Reinforce network controls for legacy applications by exploring microsegmentation strategies.
- Centralize the organizational responsibility to manage and secure:

- Core networking functions, such as cross-premises links, virtual networking, subnetting, and IP address schemes.
- Network security elements, such as virtual network appliances, cloud virtual network activity and cross-premises traffic encryption, network-based access controls, and other traditional network security components.

Reference model

If your workload requires multiple segments with shared services across each segment, start with this reference model and adapt it to your organization's needs. This model shows how functions and resources can be segmented to support an application with multiple segments.

> ⚠ **Warning**
>
> Segmentation and multiple teams suggests that your segmentation solution will expand beyond the scope of the Well Architected Framework, impacting multiple workloads consistently. When centralized teams are responsible for broad architecture decisions, like segmentation, the best practice is to begin with **Azure landing zones** in the Cloud Adoption Framework. Azure landing zones provide a conceptual architecture, reference implementations, and proven design processes to customize and implement the platform (or shared) services needed to support multiple applications. Those best practices aid in making platform wide decisions regarding **network topology and connectivity**, **segmentation** and **governance**, which should be used when these decisions impact more than one workload.



Example segments

Consider isolating shared and individual resources as shown in the preceding image.

Core Services segment

This segment hosts shared services utilized across the organization. These shared services typically include Active Directory Domain Services, DNS/DHCP, and system management tools hosted on Azure Infrastructure as a Service (IaaS) virtual machines.

Other segments

Other segments can contain grouped resources based on certain criteria. For instance, resources that are used by one specific workload or application might be contained in a separate segment. You may also segment or subsegment by lifecycle stage, like *development*, *test*, and *production*. Some resources might intersect, such as applications, and can use virtual networks for lifecycle stages.

Clear lines of responsibility

The following table describes the main functions for this reference model. See Roles, responsibilities, and permissions to learn more about permissions for the main functions.

Function	Scope	Responsibility
Policy management (Core and individual segments)	Some or all resources.	Monitor and enforce compliance with external (or internal) regulations, standards, and security policy assign appropriate permission to those roles.
Central IT operations (Core)	Across all resources.	Grant permissions to the central IT department (often the infrastructure team) to create, modify, and delete resources like virtual machines and storage.
Central networking group (Core and individual segments)	All network resources.	Centralize network management and security to reduce the potential for inconsistent strategies that create potential attacker exploitable security risks. Because all divisions of the IT and development organizations don't have the same level of network management and security knowledge and sophistication, organizations benefit from using a centralized network team's expertise and tooling. Ensure consistency and avoid technical conflicts, assign network resource responsibilities to a single central networking organization. These resources should include virtual networks, subnets, network security groups (NSG), and the virtual machines hosting virtual network appliances.

Function	Scope	Responsibility
Resource role permissions (Core)	-	For most core services, administrative privileges required are granted through the application (Active Directory, DNS/DHCP, system management tools). No other Azure resource permissions are required. If your organizational model requires these teams to manage their own VMs, storage, or other Azure resources, you can assign these permissions to those roles.
Security operations (Core and individual segments)	All resources.	Assess risk factors, identify potential mitigations, and advise organizational stakeholders who accept the risk.
IT operations (individual segments)	All resources.	Grant permission to create, modify, and delete resources. The purpose of the segment (and resulting permissions) depends on your organization structure. • Segments with resources managed by a centralized IT organization can grant the central IT department (often the infrastructure team) permission to modify these resources. • Segments managed by independent business units or functions (such as a Human Resources IT Team) can grant those teams permission to all resources in the segment. • Segments with autonomous DevOps teams don't need to grant permissions across all resources because the resource role grants permissions to application teams. For emergencies, use the service admin account (break-glass account).
Service admin (Core and individual segments)		Use the service admin role only for emergencies (and initial setup if necessary). Don't use this role for daily tasks.

Next steps

Start with this reference model and manage resources across multiple subscriptions consistently and efficiently with management groups.

Management groups

Establish segmentation with management groups

Article • 04/19/2023

Management groups can manage resources across multiple subscriptions consistently and efficiently. However, due to its flexibility, your design can become complex and compromise security and operations.

> ⓘ **Note**
>
> Management groups could be defined as part of the workload architecture. More commonly, centralized teams use management groups to consistently apply governance and segmentation strategies across multiple workloads. If a centralized team is responsible for design and operations of your management groups, reference Cloud Adoption Framework's **Azure landing zones**. The **Resource organization** design area outlines the best practice recommendations and considerations for segmentation through management groups, subscriptions, and application landing zones.

Support your segmentation strategy with management groups

Structure management groups into a simple design that guides the enterprise segmentation model.

Management groups offer the ability to consistently and efficiently manage resources (including multiple subscriptions as needed). However, because of their flexibility, it's possible to create an overly complex design. Complexity creates confusion and negatively impacts both operations and security (as illustrated by overly complex Organizational Unit (OU) and Group Policy Object (GPO) designs for Active Directory).

Microsoft recommends aligning the top level of management groups (MGs) into a simple enterprise segmentation strategy and limiting the levels to no more than two.

In the example Reference model, there are enterprise-wide resources used by all segments, a set of core services that share services, and more segments for each workload.

- Root management group for enterprise-wide resources.

Use the root management group to include identities that have the requirement to apply policies across every resource. For example, regulatory requirements, such as restrictions related to data sovereignty. This group is effective in by applying policies, permissions, tags, across all subscriptions.

> ⊗ **Caution**
>
> Be careful when using the root management group because the policies can affect all resources on Azure and potentially cause downtime or other negative impacts. For considerations, see **Use root management group with caution** later in this article.
>
> For complete guidance about using management groups for an enterprise, see **Management groups**.

- Management group for each workload segment.

  Use a separate management group for teams with limited scope of responsibility. This group is typically required because of organizational boundaries or regulatory requirements.

- Root or segment management group for the core set of services.

Use root management group with caution

Use the Root Management Group (MG) for enterprise consistency, but test changes carefully to minimize risk of operational disruption.

The root management group enables you to ensure consistency across the enterprise by applying policies, permissions, and tags across all subscriptions. Take care when you're planning and implementing assignments to the root management group. Assignments can affect every resource on Azure and potentially cause downtime or other negative impacts on productivity if there are errors or unanticipated effects.

- **Plan carefully:** Select enterprise-wide elements to the root management group that have a clear requirement to be applied across every resource or to be low impact.

  Select enterprise-wide identities that have a clear requirement to be applied across all resources. Good candidates include:

  - **Regulatory requirements** with clear business risk or impact. For example, restrictions related to data sovereignty.

    - **Near-zero potential negative impact.** For example, policy with audit effect, tag assignment, and Azure RBAC permissions assignments that have been carefully reviewed.

  Use a dedicated service principal name (SPN) to execute management group management operations, subscription management operations, and role assignment. SPN reduces the number of users who have elevated rights and follows least-privilege guidelines. Assign the **User Access Administrator** at the root management group scope (`/`) to grant the SPN access at the root level. After the SPN is granted permissions, the **User Access Administrator** role can be safely removed. In this way, only the SPN is part of the **User Access Administrator** role. Assign **Contributor** permission to the SPN, which allows tenant-level operations. This permission level ensures that you can use the SPN to deploy and manage resources to any subscription within your organization.

  Limit the number of Azure Policy assignments made at the root management group scope (`/`). This limitation minimizes debugging inherited policies in lower-level management groups.

  Don't create any subscriptions under the root management group. This hierarchy ensures that subscriptions don't only inherit the small set of Azure policies assigned at the root-level management group, as the small set of Azure policies don't represent a full set necessary for a workload.

- **Test first:** Plan, test, and validate all enterprise-wide changes on the root management group before applying (policy, tags, Azure RBAC model, and so on).

    - **Test lab:** The representative lab tenant or lab segment in the production tenant.

    - **Production pilot:** The production pilot can be a segment management group or designated subset in the subscription(s) management group.

- **Validate changes:** Validate changes to ensure they have the desired effect.

Next steps

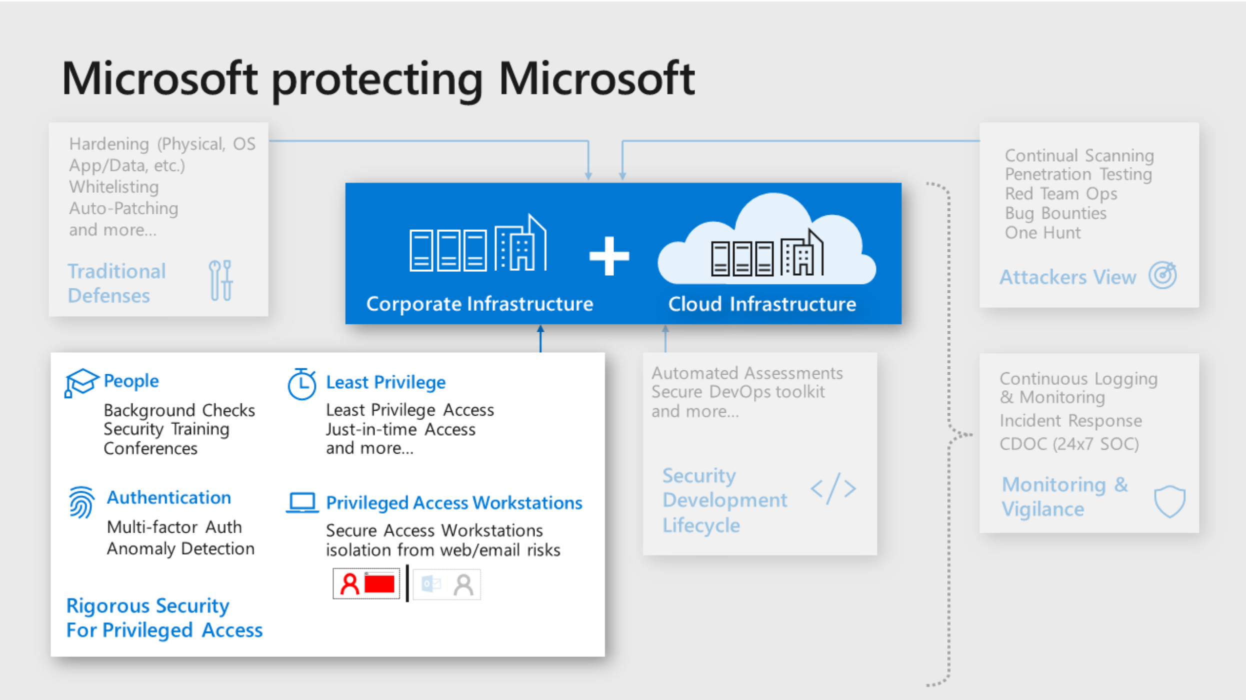
Administrative account security

Administrative account security

Article • 11/30/2022

Administration is the practice of monitoring, maintaining, and operating Information Technology (IT) systems to meet service levels that the business requires. Administration introduces some of the highest impact security risks because performing these tasks requires privileged access to a very broad set of these systems and applications. Attackers know that gaining access to an account with administrative privileges can get them access to most or all of the data they would target, making the security of administration one of the most critical security areas.

As an example, Microsoft makes significant investments in protection and training of administrators for our cloud systems and IT systems:



Microsoft's recommended core strategy for administrative privileges is to use the available controls to reduce risk.

Reduce risk exposure (scope and time): The principle of least privilege is best accomplished with modern controls that provide privileges on demand. This help to limit risk by limiting administrative privileges exposure by:

- **Scope:** *Just enough access (JEA)* provides only the required privileges for the administrative operation required (vs. having direct and immediate privileges to many or all systems at a time, which is almost never required).
- **Time:** *Just in time (JIT)* approaches provided the required privileged as they are needed.

- **Mitigate the remaining risks:** Use a combination of preventive and detective controls to reduce risks such as isolating administrator accounts from the most common risks phishing and general web browsing, simplifying and optimizing their workflow, increasing assurance of authentication decisions, and identifying anomalies from normal baseline behavior that can be blocked or investigated.

Microsoft has captured and documented best practices for protecting administrative accounts and published prioritized roadmaps for protecting privileged access that can be used as references for prioritizing mitigations for accounts with privileged access.

- Securing Privileged Access (SPA) roadmap for administrators of on-premises Active Directory
- Guidance for securing administrators of Azure Active Directory

Minimize number of critical impact admins

Grant the fewest number of accounts to privileges that can have a critical business impact.

Each admin account represents potential attack surface that an attacker can target, so minimizing the number of accounts with that privilege helps limit the overall organizational risk. Experience has taught us that membership of these privileged groups grows naturally over time as people change roles if membership not actively limited and managed.

We recommend an approach that reduces this attack surface risk while ensuring business continuity in case something happens to an administrator:

- Assign at least two accounts to the privileged group for business continuity.
- When two or more accounts are required, provide justification for each member including the original two.
- Regularly review membership & justification for each group member.

Managed accounts for admins

Ensure all critical impact admins are managed by enterprise directory to follow organizational policy enforcement.

Consumer accounts such as Microsoft accounts like @Hotmail.com, @live.com, @outlook.com, don't offer sufficient security visibility and control to ensure the organization's policies and any regulatory requirements are being followed. Because Azure deployments often start small and informally before growing into enterprise-managed tenants, some consumer accounts remain as administrative accounts long afterward for example, original Azure project managers, creating blind spots, and potential risks.

Separate accounts for admins

Ensure all critical impact admins have a separate account for administrative tasks (vs the account they use for email, web browsing, and other productivity tasks).

Phishing and web browser attacks represent the most common attack vectors to compromise accounts, including administrative accounts.

Create a separate administrative account for all users that have a role requiring critical privileges. For these administrative accounts, block productivity tools like Office 365 email (remove license). If possible, block arbitrary web browsing (with proxy and/or application controls) while allowing exceptions for browsing to the Azure portal and other sites required for administrative tasks.

No standing access / just in time privileges

Avoid providing permanent "standing" access for any critical impact accounts.

Permanent privileges increase business risk by increasing the time an attacker can use the account to do damage. Temporary privileges force attackers targeting an account to either work within the limited times the admin is already using the account or to initiate privilege elevation (which increases their chance of being detected and removed from the environment).

Grant privileges required only as required using one of these methods:

- **Just in time:** Enable Azure AD Privileged Identity Management (PIM) or a third party solution to require following an approval workflow to obtain privileges for critical impact accounts.

- **Break glass:** For rarely used accounts, follow an emergency access process to gain access to the accounts. This is preferred for privileges that have little need for regular operational usage like members of global admin accounts.

Emergency access or 'Break Glass' accounts

Ensure you have a mechanism for obtaining administrative access in case of an emergency.

While rare, sometimes extreme circumstances arise where all normal means of administrative access are unavailable.

We recommend following the instructions at Managing emergency access administrative accounts in Azure AD and ensure that security operations monitor these accounts carefully.

Admin workstation security

Ensure critical impact admins use a workstation with elevated security protections and monitoring.

Attack vectors that use browsing and email like phishing are cheap and common. Isolating critical impact admins from these risks will significantly lower your risk of a major incident where one of these accounts is compromised and used to materially damage your business or mission.

Choose level of admin workstation security based on the options available at https://aka.ms/securedworkstation

- **Highly Secure Productivity Device (Enhanced Security Workstation or Specialized Workstation)**
  You can start this security journey for critical impact admins by providing them with a higher security workstation that still allows for general browsing and productivity tasks. Using this as an interim step helps ease the transition to fully isolated workstations for both the critical impact admins as well as the IT staff supporting these users and their workstations.

- **Privileged Access Workstation (Specialized Workstation or Secured Workstation)**
  These configurations represent the ideal security state for critical impact admins as they heavily restrict access to phishing, browser, and productivity application attack vectors. These workstations don't allow general internet browsing, only allow browser access to Azure portal and other administrative sites.

Critical impact admin dependencies – Account/Workstation

Carefully choose the on-premises security dependencies for critical impact accounts and their workstations.

To contain the risk from a major incident on-premises spilling over to become a major compromise of cloud assets, you must eliminate or minimize the means of control that on premises resources have to critical impact accounts in the cloud. As an example, attackers who compromise the on-premises Active Directory can access and compromise cloud-based assets that rely on those accounts like resources in Azure, Amazon Web Services (AWS), ServiceNow, and so on. Attackers can also use workstations joined to those on premises domains to gain access to accounts and services managed from them.

Choose the level of isolation from on premises means of control also known as security dependencies for critical impact accounts.

- **User Accounts:** Choose where to host the critical impact accounts
  - Native Azure AD Accounts - Create Native Azure AD Accounts that are not synchronized with on-premises active directory.
  - Synchronize from On Premises Active Directory.
  - Use existing accounts hosted in the on-premises active directory.
- **Workstations:** Choose how you will manage and secure the workstations used by critical admin accounts:
  - Native Cloud Management and Security (Recommended): Join workstations to Azure AD & Manage/Patch them with Intune or other cloud services. Protect and Monitor with Windows Microsoft Defender for Endpoint or another cloud service not managed by on premises based accounts.
  - Manage with Existing Systems: Join existing AD domain and use existing management/security.

Passwordless or multifactor authentication for admins

Require all critical impact admins to use passwordless authentication or multifactor authentication (MFA).

Attack methods have evolved to the point where passwords alone cannot reliably protect an account. This is well documented in a Microsoft Ignite Session.

Administrative accounts and all critical accounts should use one of the following methods of authentication. These capabilities are listed in preference order by highest cost/difficulty to attack (strongest/preferred options) to lowest cost/difficult to attack:

- Passwordless (such as Windows Hello)
- Passwordless (Authenticator App)
- Multifactor Authentication

Note that SMS Text Message based MFA has become very inexpensive for attackers to bypass, so we recommend you avoid relying on it. This option is still stronger than passwords alone, but is much weaker than other MFA options.

Enforce conditional access for admins - Zero Trust

Authentication for all admins and other critical impact accounts should include measurement and enforcement of key security attributes to support a Zero Trust strategy.

Attackers compromising Azure Admin accounts can cause significant harm. Conditional Access can significantly reduce that risk by enforcing security hygiene before allowing access to Azure management.

Configure Conditional Access policy for Azure management that meets your organization's risk appetite and operational needs.

- Require Multifactor Authentication and/or connection from designated work network.
- Require Device **integrity with Microsoft Defender for Endpoint** (Strong Assurance).

Avoid granular and custom permissions

Avoid permissions that specifically reference individual resources or users.

Specific permissions create unneeded complexity and confusion as they don't carry the intention to new similar resources. This then accumulates into a complex legacy configuration that is difficult to maintain or change without fear of "breaking something" – negatively impacting both security and solution agility.

Instead of assigning specific resource-specific permissions, use either:

- Management Groups for enterprise-wide permissions.
- Resource groups for permissions within subscriptions.

Instead of granting permissions to specific users, assign access to groups in Azure AD. If there isn't an appropriate group, work with the identity team to create one. This allows you to add and remove group members externally to Azure and ensure permissions are current, while also allowing the group to be used for other purposes such as mailing lists.

Use built-in roles

Use built-in roles for assigning permissions where possible.

Customization leads to complexity that increases confusion and makes automation more complex, challenging, and fragile. These factors all negatively impact security.

We recommend that you evaluate the built-in roles designed to cover most normal scenarios. Custom roles are a powerful and sometimes useful capability, but they should be reserved for cases when built in roles won't work.

Establish lifecycle management for critical impact accounts

Ensure you have a process for disabling or deleting administrative accounts when admin personnel leave the organization (or leave administrative positions).

See Regularly review critical access for more details.

Attack simulation for critical impact accounts

Regularly simulate attacks against administrative users with current attack techniques to educate and empower them.

People are a critical part of your defense, especially your personnel with access to critical impact accounts. Ensuring these users (and ideally all users) have the knowledge and skills to avoid and resist attacks will reduce your overall organizational risk.

You can use Office 365 Attack Simulation capabilities or any number of third party offerings.

Azure identity and access management considerations

Article • 04/19/2023

Most architectures have shared services that are hosted and accessed across networks. Those services share common infrastructure and users need to access resources and data from anywhere. For such architectures, a common way to secure resources is to use network controls. However, that isn't enough.

Provide security assurance through *identity management*: the process of authenticating and authorizing security principals. Use identity management services to authenticate and grant permission to users, partners, customers, applications, services, and other entities.

> **Note**
>
> Identity management is typically a centralized function not controlled by the workload team as a part of the workload's architecture. Unless the workload team is responsible for a dedicated identity store, the guidance in the **Azure identity and access management design area** of the Cloud Adoption Framework should be referenced when implementing identity solutions which support multiple workloads.

Checklist

How are you managing the identity for your workload?

- ✓ Define clear lines of responsibility and separation of duties for each function. Restrict access based on a need-to-know basis and least privilege security principles.
- ✓ Assign permissions to users, groups, and applications at a certain scope through Azure RBAC. Use built-in roles when possible.
- ✓ Prevent deletion or modification of a resource, resource group, or subscription through management locks.
- ✓ Use managed identities to access resources in Azure.
- ✓ Support a single enterprise directory. Keep the cloud and on-premises directories synchronized, except for critical-impact accounts.
- ✓ Set up Azure AD Conditional Access. Enforce and measure key security attributes when authenticating all users, especially for critical-impact accounts.

- ✓ Have a separate identity source for non-employees.
- ✓ Preferably use passwordless methods or opt for modern password methods.
- ✓ Block legacy protocols and authentication methods.

Azure security benchmark

The Azure Security Benchmark includes a collection of high-impact security recommendations you can use to help secure the services you use in Azure:

> The questions in this section are aligned to the Azure Security Benchmarks Identity and Access Control.

Azure services for identity

The considerations and best practices in this section are based on these Azure services:

- Azure AD
- Azure AD B2B
- Azure AD B2C

Reference architecture

Here are some reference architectures related to identity and access management:

Integrate on-premises AD domains with Azure AD

Integrate on-premises AD with Azure

Next steps

Monitor the communication between segments. Use data to identify anomalies, set alerts, or block traffic to mitigate the risk of attackers crossing segmentation boundaries.

Network security

Related links

Five steps to securing your identity infrastructure

> Go back to the main article: Overview of the security pillar

Roles, responsibilities, and permissions

Article • 04/19/2023

In an organization, several teams work together to make sure that the workload and the supporting infrastructure are secure. To avoid confusion that can create security risks, define clear lines of responsibility and separation of duties.

Based on Microsoft's experience with many cloud adoption projects, establishing clearly defined roles and responsibilities for specific functions in Azure avoids confusion that can lead to human and automation errors creating security risk.

Clear lines of responsibility

Do the teams have a clear view on responsibilities and individual/group access levels?

Designate the parties responsible for specific functions in Azure.

Clearly documenting and sharing the contacts responsible for each of these functions creates consistency and facilitates communication. Based on Microsoft's experience with many cloud adoption projects, consistency and communication prevent confusion that can lead to human and automation errors that create security risks.

Designate groups (or individual roles) that are responsible for key functions.

> **Note**
>
> A centralized team might be responsible for establishing these roles across your organization to provide consistent support across all workload teams. If your team is not solely responsible for each of the following roles, consult the guidance on **Aligning responsibilities across teams** with a focus on understanding how your workload team will interface with a **Cloud platform team**, **Central IT**, or a **Cloud center of excellence**.

Group or individual role	Responsibility
Network Security	*Typically an existing network security team.* Configuration and maintenance of Azure Firewall, Network Virtual Appliances (and associated routing), Web Application Firewall (WAF), network security groups, application security groups (ASG), and other cross-network traffic.

Group or individual role	Responsibility
Network Management	*Typically an existing network operations team.* Enterprise-wide virtual network and subnet allocation.
Server Endpoint Security	*Typically IT operations, security, or jointly.* Monitor and remediate server security (patching, configuration, endpoint security).
Incident Monitoring and Response	*Typically a security operations team.* Incident monitoring and response to investigate and remediate security incidents in security information and event management (SIEM) or source console such as Microsoft Defender for Cloud Azure AD Identity Protection.
Policy Management	*Typically a GRC team and an architecture team.* Apply governance based on risk analysis and compliance requirements. Set direction for use of Azure role-based access control (Azure RBAC), Microsoft Defender for Cloud, administrator protection strategy, and Azure Policy to govern Azure resources.
Identity Security and Standards	*Typically a security team and an identity team jointly.* Set direction for Azure AD directories, PIM/PAM usage, multifactor authentication, password and synchronization configuration, and application identity standards.

> ⓘ Note
>
> Application roles and responsibilities should cover different access level of each operational function. For example, publish production release, access customer data, manipulate database records, and so on. Application teams should include central functions listed in the preceding table.

Assign permissions

Grant roles the appropriate permissions that start with least privilege and add more based on your operational needs. Provide clear guidance to your technical teams that implement permissions. This clarity makes it easier to detect and correct that reduces human errors such as overpermissioning.

> ⓘ Note
>
> Many organizations manage identity, access, and permissions from a centralized cloud platform team using Cloud Adoption Framework Azure landing zones as a guide for configuration and permissions across multiple workloads. If identity and

access are managed outside of the workload team, see the **Identity and access management design area** to understand how to apply the proper level of permissions in your application's landing zone.

- Assign permissions at the management group level for the segment instead of individual subscriptions. Assigning permissions drives consistency and ensures application to future subscriptions. In general, avoid granular and custom permissions.
- Consider the built-in roles in Azure before creating custom roles to grant the appropriate permissions to VMs and other objects.
- **Security managers** group membership might be appropriate for smaller teams or organizations where security teams have extensive operational responsibilities.

When assigning permissions for a segment, consider consistency while allowing flexibility to accommodate several organizational models. These models can range from a single centralized IT group to mostly independent IT and DevOps teams.

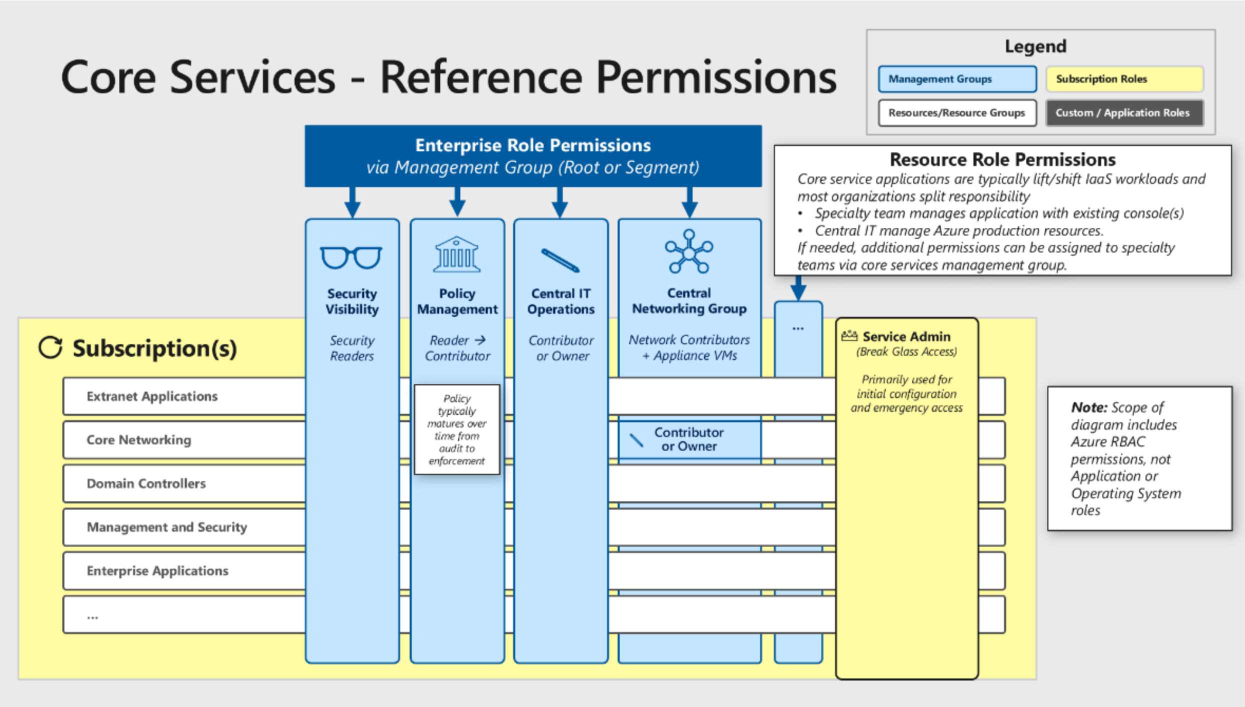
Reference model example

This section uses a reference model to demonstrate the considerations for assigning permissions for different segments. If your workload architecture requires segmentation and shared services spanning multiple segments for the same workload, Microsoft recommends starting from these models and adapting to your organization.

> ⚠ **Warning**
>
> Shared services are seldom deployed or managed as part of a single workload or by the workload team. When centralized teams provide shared service, the best practice is to begin with **Azure landing zones** in the Cloud Adoption Framework. Azure Landing Zones provide a conceptual architecture, reference implementations, and proven design processes to customize and implement the platform (or shared) services needed to support multiple applications. Those best practices aid in making platform wide decisions regarding **network topology and connectivity**, **segmentation** and **governance**, which should be used when these decisions impact more than one workload.

Core services reference permissions

This segment hosts shared services utilized across the organization. These shared services typically include Active Directory Domain Services, DNS/DHCP, System Management Tools hosted on Azure Infrastructure as a Service (IaaS) virtual machines.



Security visibility across all resources: For security teams, grant read-only access to security attributes for all technical environments. This access level is needed to assess risk factors, identify potential mitigations, and advise organizational stakeholders who accept the risk. For more information, see Security team visibility.

Policy management across some or all resources: To monitor and enforce compliance with external (or internal) regulations, standards, and security policy, assign appropriate permission to those roles. The roles and permissions you choose depend on the organizational culture and expectations of the policy program. See Microsoft Cloud Adoption Framework for Azure.

Before defining the policies, consider:

- How is the organization's security audited and reported? Is there mandatory reporting?
- Are the existing security practices working?
- Are there any requirements specific to industry, government, or regulatory requirements?

Designate group(s) (or individual roles) for central functions that affect shared services and applications.

After the policies are set, continuously improve those standards incrementally. Make sure that the security posture doesn't degrade over time by having auditing and monitoring compliance. For information about managing security standards of an organization, see governance, risk, and compliance (GRC).

Central IT operations across all resources: Grant permissions to the central IT department (often the infrastructure team) to create, modify, and delete resources like virtual machines and storage. **Contributor** or **owner** roles are appropriate for this function.

Central networking group across network resources: To ensure consistency and avoid technical conflicts, assign network resource responsibilities to a single central networking organization. These resources should include virtual networks, subnets, Network Security Groups (NSG), and the virtual machines hosting virtual network appliances. Assign network resource responsibilities to a single central networking organization. The **network contributor** role is appropriate for this group. For more information, see Centralize network management and security.

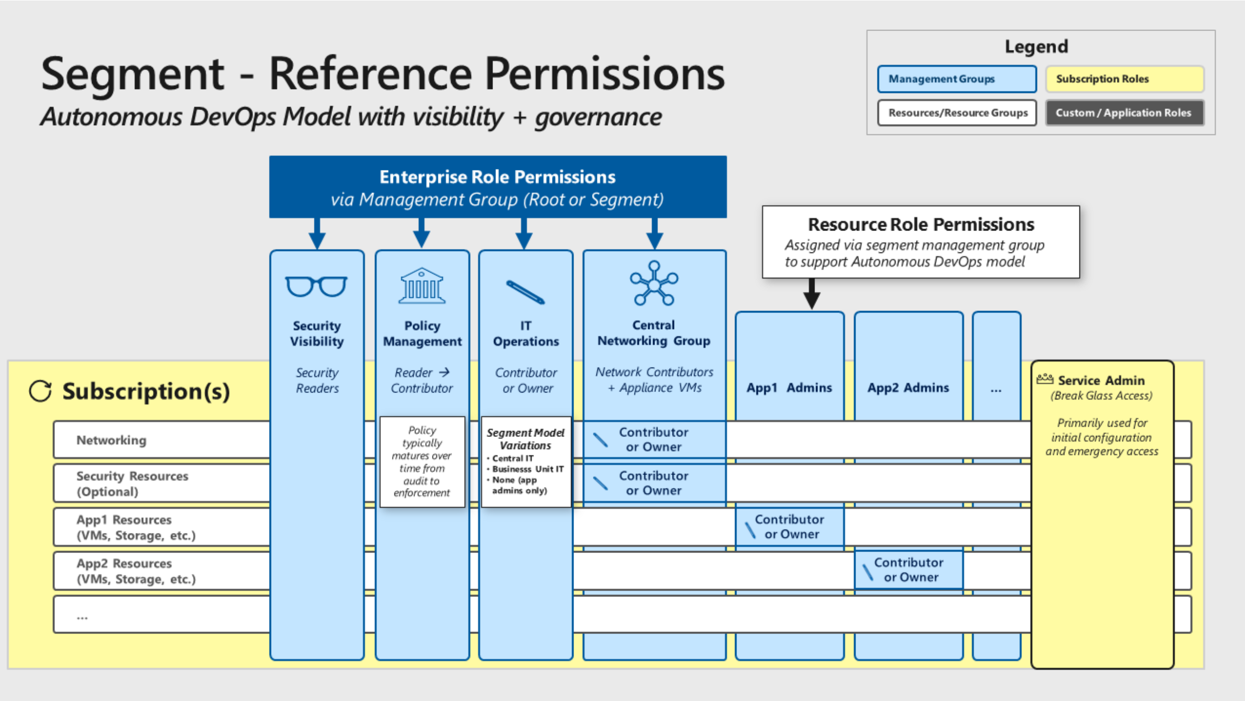
Resource role permissions: For most core services, administrative privileges required to manage them are granted through the application (Active Directory, DNS/DHCP, System Management Tools), so no other Azure resource permissions are required. If your organizational model requires these teams to manage their own VMs, storage, or other Azure resources, you can assign these permissions to those roles.

Workload segments with autonomous DevOps teams manage the resources associated with each application. The actual roles and permissions depend on the application size and complexity, the application team's size and complexity, and the organization and application team's culture.

Service admin (break glass account): Use the **service administrator** role only for emergencies and initial setup. Don't use this role for daily tasks. For more information, see Emergency access ('break glass' accounts).

Segment reference permissions

This segment permission design provides consistency while allowing enough flexibility to accommodate the range of organizational models. The organizational models span from a single centralized IT group to mostly independent IT and DevOps teams.



Security visibility across all resources: For security teams, grant read-only access to security attributes for all technical environments. This access level is needed to assess risk factors, identify potential mitigations, and advise organizational stakeholders who accept the risk. See Security Team Visibility.

Policy management across some or all resources: To monitor and enforce compliance with external (or internal) regulations, standards, and security policy assign appropriate permission to those roles. The roles and permissions you choose depend on the organizational culture and expectations of the policy program. See Security Baseline discipline overview.

IT Operations across all resources: Grant permission to create, modify, and delete resources. The purpose of the segment (and resulting permissions) depends on your organization structure.

- Segments with resources managed by a centralized IT organization can grant the central IT department (often the infrastructure team) permission to modify these resources.
- Segments managed by independent business units or functions (such as a Human Resources IT Team) can grant those teams permission to all resources in the segment.
- Segments with autonomous DevOps teams don't need to grant permissions across all resources because the resource role grants permissions to application teams. For emergencies, use the service admin account (break-glass account).

Central networking group across network resources: To ensure consistency and avoid technical conflicts, assign network resource responsibilities to a single central networking organization. These resources should include virtual networks, subnets, Network Security Groups (NSG), and the virtual machines hosting virtual network appliances. See Centralize Network Management And Security.

Resource role permissions: Segments with autonomous DevOps teams manage the resources associated with each application. The actual roles and permissions depend on the application size and complexity, the application team's size and complexity, and the organization and application team's culture.

Service admin (break glass account): Use the service admin role only for emergencies (and initial setup if necessary). Don't use this role for daily tasks. For more information, see Emergency access or 'Break Glass' accounts).

Security team visibility

An application team needs to be aware of security initiatives to align their security improvement plans with the outcome of those activities. Provide security teams read-only access to the security aspects of all technical resources in their purview.

Security organizations need visibility into the technical environment to perform their duties of assessing and reporting on organizational risk. Without this visibility, security has to rely on information provided from groups operating in the environment, which have potential conflict of interest (and other priorities).

Security teams might separately be granted other privileges if they have operational responsibilities or a requirement to enforce compliance on Azure resources.

For example in Azure, assign security teams to the **security readers** permission that provides access to measure security risk (without providing access to the data itself).

For enterprise security groups with broad responsibility for security of Azure, you can assign this permission using:

- *Root management group* – for teams responsible for assessing and reporting risk on all resources
- *Segment management group(s)* – for teams with limited scope of responsibility (typically required because of organizational boundaries or regulatory requirements)

ⓘ **Important**

Because security will have broad access to the environment (and visibility into potentially exploitable vulnerabilities), treat security teams as critical impact accounts and apply the same protections as administrators. The **Administrative account security** section details these controls for Azure.

Suggested actions

- Define a process for aligning communication, investigation, and hunting activities with the application team.
- Following the principle of least privilege, establish access control to all cloud environment resources for security teams with sufficient access. Security teams gain the required visibility into the technical environment and to perform their duties of assessing, and reporting on organizational risk.

Learn more

Engage your organization's security team

Manage connected tenants

Does your security team have visibility into all existing subscriptions and cloud environments? How do they discover new ones?

Ensure your security organization is aware of all enrollments and associated subscriptions connected to your existing environment (via ExpressRoute or a site-to-site VPN) and monitoring as part of the overall enterprise.

These Azure resources are part of your enterprise environment and security organizations require visibility into them. Security organizations need this access to assess risk and to identify whether organizational policies and applicable regulatory requirements are being followed.

The organizations' cloud infrastructure should be well documented, with security team access to all resources required for monitoring and insight. Frequent scans of the cloud-connected assets should be performed to ensure no other subscriptions or tenants have been added outside of organizational controls. Regularly review Microsoft guidance to ensure security team access best practices are consulted and followed.

Suggested actions

Ensure all Azure environments that connect to your production environment and network apply your organization's policy, and IT governance controls for security.

You can discover existing connected tenants using a tool provided by Microsoft for guidance on permissions.

Next steps

Restrict access to Azure resources based on a need-to-know basis starting with the principle of least privilege security and add more based on your operational needs.

Azure control plane security

Related links

For considerations about using management groups to reflect the organization's structure within an Azure Active Directory (Azure AD) tenant, see Management groups.

> Back to the main article: Azure identity and access management considerations

Azure control plane security

Article • 11/30/2022

The term *control plane* refers to the management of resources in your subscription. These activities include creating, updating, and deleting Azure resources as required by the technical team.

Azure Resource Manager handles all control plane requests and applies restrictions that you specify through Azure role-based access control (Azure RBAC), Azure Policy, locks. Apply those restrictions based on the requirement of the organization.

It's recommended to implement Infrastructure as Code, and to deploy application infrastructure through automation, and CI/CD for consistency and auditing purposes.

Key points

- ✓ Restrict access based on a need-to-know basis and least privilege security principles.
- ✓ Assign permissions to users, groups, and applications at a certain scope through Azure RBAC.
- ✓ Use built-in roles when possible.
- ✓ Prevent deletion or modification of a resource, resource group, or subscription through management locks.
- ✓ Use less critical control in your CI/CD pipeline for development and test environments.

Roles and permission assignment

Is the workload infrastructure protected with Azure role-based access control (Azure RBAC)?

Azure role-based access control (Azure RBAC) provides the necessary tools to maintain separation of concerns for administration and access to application infrastructure. Decide who has access to resources at the granular level and what they can do with those resources. For example:

- Developers can't access production infrastructure.
- Only the SecOps team can read and manage Key Vault secrets.
- If there are multiple teams, Project A team can access and manage Resource Group A and all resources within.

> Grant roles the appropriate permissions that start with least privilege and add more based on your operational needs. Provide clear guidance to your technical teams that implement permissions. This clarity makes it easier to detect and correct which reduces human errors such as overpermissioning.

Azure RBAC helps you manage that separation. You can assign permissions to users, groups, and applications at a certain scope. The scope of a role assignment can be a subscription, a resource group, or a single resource. For details, see Azure role-based access control (Azure RBAC).

- Assign permissions at management group instead of individual subscriptions to drive consistency and ensure application to future subscriptions.
- Consider the built-in roles before creating custom roles to grant the appropriate permissions to resources and other objects.

For example, assign security teams with the **Security Readers** permission that provides access needed to assess risk factors, identify potential mitigations, without providing access to the data.

> ⓘ **Important**
>
> Treat security teams as critical accounts and apply the same protections as administrators.

Learn more

Azure RBAC documentation

Management locks

Are there resource locks applied on critical parts of the infrastructure?

Unlike Azure role-based access control, management locks are used to apply a restriction across all users and roles.

Critical infrastructure typically doesn't change often. Use management locks to prevent deletion or modification of a resource, resource group, or subscription. Lock in use cases where only specific roles and users with permissions can delete, or modify resources.

As an administrator, you may need to lock a subscription, resource group, or resource to prevent other users in your organization from accidentally deleting or modifying critical resources. You can set the lock level to `CanNotDelete` or `ReadOnly`. In the portal, the locks are called **Delete** and **Read-only**, respectively:

- *CanNotDelete* means authorized users can still read and modify a resource, but they can't delete the resource.
- *ReadOnly* means authorized users can read a resource, but they can't delete or update the resource. Applying this lock is similar to restricting all authorized users to the permissions granted by the *Reader* role.

When you apply a lock at a parent scope, all resources within that scope inherit the same lock. Even resources you add later inherit the lock from the parent. The most restrictive lock in the inheritance takes precedence.

Unlike role-based access control, you use management locks to apply a restriction across all users and roles. To learn about setting permissions for users and roles, see Azure role-based access control (Azure RBAC).

Identify critical infrastructure and evaluate resource lock suitability.

Set locks in the DevOps process carefully because modification locks can sometimes block automation. For examples of those blocks and considerations, see Considerations before applying locks.

Suggested actions

- Restrict application infrastructure access to CI/CD only.
- Use conditional access policies to restrict access to Microsoft Azure Management.
- Configure role-based and resource-based authorization within Azure AD.

Learn more

- Manage access to Azure management with Conditional Access
- Role-based and resource-based authorization

Next steps

Grant or deny access to a system by verifying whether the accessor has the permissions to perform the requested action.

Authentication

Related links

> Back to the main article: Azure identity and access management considerations

Authentication with Azure AD

Article • 11/30/2022

Authentication is a process that grants or denies access to a system by verifying the accessor's identity. Use a managed identity service for all resources to simplify overall management (such as password policies) and minimize the risk of oversights or human errors. Azure Active Directory (Azure AD) is the one-stop-shop for identity and access management service for Azure.

Key points

- Use Managed Identities to access resources in Azure.
- Keep the cloud and on-premises directories synchronized, except for high-privilege accounts.
- Preferably use passwordless methods or opt for modern password methods.
- Enable Azure AD conditional access based on key security attributes when authenticating all users, especially for high-privilege accounts.

Use identity-based authentication

How is the application authenticated when communicating with Azure platform services?

Managed identities enable Azure Services to authenticate to each other without presenting explicit credentials via code.

Managed identities for Azure resources is a feature of Azure Active Directory. Each of the Azure services that support managed identities for Azure resources are subject to their own timeline. Make sure you review the availability status of managed identities for your resource and known issues before you begin. The feature provides Azure services with an automatically managed identity in Azure AD. You can use the identity to authenticate to any service that supports Azure AD authentication, including Key Vault, without any credentials in your code. The managed identities for Azure resources feature is free with Azure AD for Azure subscriptions, there's no additional cost.

There are two types of managed identities:

- A system-assigned managed identity is enabled directly on an Azure service instance. When the identity is enabled, Azure creates an identity for the instance in the Azure AD tenant that's trusted by the subscription of the instance. After the

identity is created, the credentials are provisioned onto the instance. The life cycle of a system-assigned identity is directly tied to the Azure service instance that it's enabled on. If the instance is deleted, Azure automatically cleans up the credentials and the identity in Azure AD.
- A user-assigned managed identity is created as a standalone Azure resource. Through a create process, Azure creates an identity in the Azure AD tenant that's trusted by the subscription in use. After the identity is created, the identity can be assigned to one or more Azure service instances. The life cycle of a user-assigned identity is managed separately from the life cycle of the Azure service instances to which it's assigned.

Authenticate with identity services instead of cryptographic keys. On Azure, Managed Identities eliminate the need to store credentials that might be leaked inadvertently. When Managed Identity is enabled for an Azure resource, it's assigned an identity that you can use to obtain Azure AD tokens. For more information, see Azure AD-managed identities for Azure resources.

For example, an Azure Kubernetes Service (AKS) cluster needs to pull images from Azure Container Registry (ACR). To access the image, the cluster needs to know the ACR credentials. The recommended way is to enable Managed Identities during cluster configuration. That configuration assigns an identity to the cluster and allows it to obtain Azure AD tokens.

This approach is secure because Azure handles the management of the underlying credentials for you.

- The identity is tied to the lifecycle of the resource, in the AKS cluster example. When the resource is deleted, Azure automatically deletes the identity.
- Azure AD manages the timely rotation of secrets for you.

> **Tip**
>
> Here are the resources for the preceding example:
>
> **GitHub: Azure Kubernetes Service (AKS) Secure Baseline Reference Implementation**.
>
> The design considerations are described in **Azure Kubernetes Service (AKS) production baseline**.

Suggested actions

- Review workload authentication and identify opportunities to convert explicit credentials (for example, connection string and API key) to use managed identities.
- For all new Azure workloads, standardize on using managed identities where applicable.

Learn more

What are managed identities for Azure resources?

What kind of authentication is required by application APIs?

Don't assume that API URLs used by a workload are hidden and can't get exposed to attackers. For example, JavaScript code on a website can be viewed. A mobile application can be decompiled and inspected. Even for internal APIs used only on the backend, a requirement of authentication can increase the difficulty of lateral movement if an attacker gets network access. Typical mechanisms include API keys, authorization tokens and IP restrictions.

Managed Identity can help an API be more secure because it replaces the use of human-managed service principals and can request authorization tokens.

How is user authentication handled in the application?

Don't use custom implementations to manage user credentials. Instead, use Azure AD or other managed identity providers such as Microsoft account Azure B2C. Managed identity providers provide additional security features such as modern password protections, multifactor authentication (MFA), and resets. In general, passwordless protections are preferred. Also, modern protocols like OAuth 2.0 use token-based authentication with limited timespan.

Are authentication tokens cached securely and encrypted when sharing across web servers?

Application code should first try to get OAuth access tokens silently from a cache before attempting to acquire a token from the identity provider, to optimize performance and maximize availability. Tokens should be stored securely and handled as any other credentials. When there's a need to share tokens across application servers (instead of each server acquiring and caching their own) encryption should be used.

For information, see Acquire and cache tokens.

Choose a system with cross-platform support

Use a single identity provider for authentication on all platforms (operating systems, cloud providers, and third-party services.

Azure AD can be used to authenticate Windows, Linux, Azure, Office 365, other cloud providers, and third-party services as service providers.

For example, improve the security of Linux virtual machines (VMs) in Azure with Azure AD integration. For details, see Log in to a Linux virtual machine in Azure using Azure Active Directory authentication.

Centralize all identity systems

Keep your cloud identity synchronized with the existing identity systems to ensure consistency and reduce human errors.

Consistency of identities across cloud and on-premises will reduce human error and resulting security risk. Teams managing resources in both environments need a consistent authoritative source to achieve security assurances. For monitoring, if identity can be determined without an intermediate mapping process, security efficiency is improved.

Synchronization is all about providing users an identity in the cloud based on their on-premises identity. Whether or not they will use synchronized account for authentication or federated authentication, the users will still need to have an identity in the cloud. This identity will need to be maintained and updated periodically. The updates can take many forms, from title changes to password changes.

Start by evaluating the organization's on-premises identity solution and user requirements. This evaluation is important, as it defines the technical requirements for how user identities will be created and maintained in the cloud. For the majority of organizations, Active Directory is established on-premises and will be the on-premises directory from which users will be synchronized, but this is not always the case.

Consider using Azure AD Connect for synchronizing Azure AD with your existing on-premises directory. For migration projects, have a requirement to complete this task before an Azure migration and development projects begin.

> ⓘ **Important**
>
> Don't synchronize high-privilege accounts to an on-premises directory. If an attacker gets full control of on-premises assets, they can compromise a cloud

account. This strategy will limit the scope of an incident. For more information, see **Critical impact account dependencies**.

Synchronization is blocked by default in the default Azure AD Connect configuration. Make sure that you haven't customized this configuration. For information about filtering in Azure AD, see **Azure AD Connect sync: Configure filtering**.

For more information, see hybrid identity providers.

Tip

Here are the resources for the preceding example::

The design considerations are described in **Integrate on-premises Active Directory domains with Azure AD**.

Learn more

Synchronize the hybrid identity systems

Use passwordless authentication

Attackers constantly scan public cloud IP ranges for open management ports. They attempt to exploit weak credentials (*password spray*) and unpatched vulnerabilities in management protocols like SSH, and RDP. Preventing direct internet access to virtual machines stops a misconfiguration or oversight becoming more serious.

Attack methods have evolved to the point where passwords alone cannot reliably protect an account. Modern authentication solutions including passwordless and multifactor authentication increase security posture through strong authentication.

Remove the use of passwords, when possible. Also, require the same set of credentials to sign in and access the resources on-premises or in the cloud. This requirement is crucial for accounts that require passwords, such as admin accounts.

With modern authentication and security features in Azure AD, that basic password should be supplemented or replaced with more secure authentication methods. Each organization has different needs when it comes to authentication. Microsoft offers the following three passwordless authentication options that integrate with Azure Active Directory (Azure AD):

- Windows Hello for Business
- Microsoft Authenticator app
- FIDO2 security keys

It's recommended to follow a four-stage plan to become passwordless:

- Develop password replacement offering
- Reduce user-visible password surface area
- Transition into password-less deployment
- Eliminate passwords from the identity directory

The following methods of authentication are ordered by highest cost/difficulty to attack (strongest/preferred options) to lowest cost/difficult to attack:

- Passwordless authentication. Some examples of this method include Windows Hello or Authenticator App.
- MFA. Although this method is more effective than passwords, we recommend that you avoid relying on SMS text message-based MFA. For more information, see Enable per-user Azure Active Directory MFA to secure sign-in events.
- Managed Identities. See Use identity-based authentication.

Those methods apply to all users, but should be applied first and strongest to accounts with administrative privileges.

An implementation of this strategy is enabling single sign-on (SSO) to devices, apps, and services. By signing in once using a single user account, you can grant access to all the applications and resources per business needs. Users don't have to manage multiple sets of usernames and passwords. You can provision or de-provision application access automatically. For more information, see Single sign-on.

Suggested actions

- Develop a passwordless strategy that requires MFA for all users without significantly impacting operations.
- Ensure policy and processes require restricting, and monitoring direct internet connectivity by virtual machines.

Learn more

- Passwordless Strategy
- Remove Virtual Machine (VM) direct internet connectivity

Use modern password protection

Require modern protections through methods that reduce the use of passwords. Modern authentication protocols support strong controls such as MFA and should be used instead of legacy authentication methods. Use of legacy methods increases risk of credential exposure.

Modern authentication is a method of identity management that offers more secure user authentication and authorization. It's available for Office 365 hybrid deployments of Skype for Business server on-premises and Exchange server on-premises, and split-domain Skype for Business hybrids.

Modern authentication is an umbrella term for a combination of authentication and authorization methods between a client (for example, your laptop or your phone) and a server, as well as some security measures that rely on access policies that you may already be familiar with. It includes:

- *Authentication methods*: MFA; smart card authentication; client certificate-based authentication
- *Authorization methods*: Microsoft's implementation of Open Authorization (OAuth)
- *Conditional access policies*: Mobile Application Management (MAM) and Azure Active Directory (Azure AD) Conditional Access

Review workloads that do not leverage modern authentication protocols and convert where possible. In addition, standardize using modern authentication protocols for all future workloads.

For Azure, enable protections in Azure AD:

1. Configure Azure AD Connect to synchronize password hashes. For information, see Implement password hash synchronization with Azure AD Connect sync.

2. Choose whether to automatically or manually remediate issues found in a report. For more information, see Monitor identity risks.

For more information about supporting modern passwords in Azure AD, see the following articles:

- What is Identity Protection?
- Enforce on-premises Azure AD Password Protection for Active Directory Domain Services
- Users at risk security report
- Risky sign-ins security report

For more information about supporting modern passwords in Office 365, see the following article:

What is modern authentication?

Enable conditional access

Modern cloud-based applications are typically accessible over the internet, making network location-based access inflexible and single-factor passwords a liability. Conditional access describes your authentication policy for an access decision. For example, if a user is connecting from an InTune-managed corporate PC, they might not be challenged for MFA every time, but if the user suddenly connects from a different device in a different geography, MFA is required.

Grant access requests based on the requestors' trust level and the target resources' sensitivity.

Are there any conditional access requirements for the application?

Workloads can be exposed over public internet and location-based network controls are not applicable. To enable conditional access, understand what restrictions are required for the use case. For example, MFA is a necessity for remote access; IP-based filtering can be used to enable adhoc debugging (VPNs are preferred).

Configure Azure AD Conditional Access by setting up Access policy for Azure management based on your operational needs. For information, see Manage access to Azure management with Conditional Access.

Conditional access can be an effective way to phase out legacy authentication and associated protocols. The policies must be enforced for all admins and other critical impact accounts. Start by using metrics and logs to determine users who still authenticate with old clients. Next, disable any down-level protocols that aren't used, and set up conditional access for all users who aren't using legacy protocols. Finally, give notice and guidance to users about upgrading before blocking legacy authentication completely. For more information, see Azure AD Conditional Access support for blocking legacy auth.

Suggested actions

Implement conditional access policies for this workload.

Learn more about Azure AD Conditional Access.

Next

Grant or deny access to a system by verifying the accessor's identity.

Authorization

Related links

> Back to the main article: Azure identity and access management considerations

Authorization with Azure AD

Article • 11/30/2022

Authorization is a process that grants or denies access to a system by verifying whether the accessor has the permissions to perform the requested action. The accessor in this context is the workload (cloud application) or the user of the workload. The action might be operational or related to resource management. There are two main approaches to authorization: role-based and resource-based. Both can be configured with Azure AD.

Key points

- Use a mix of role-based and resource-based authorization. Start with the principle of least privilege and add more actions based on your needs.
- Define clear lines of responsibility and separation of duties for application roles and the resources it can manage. Consider the access levels of each operational function, such as permissions needed to publish production release, access customer data, manipulate database records.
- Do not provide permanent access for any critical accounts. Elevate access permissions that are based on approval and is time bound using Azure AD Privileged Identity Management (Azure AD PIM).|

Role-based authorization

This approach authorizes an action based on the role assigned to a user. For example, some actions require an administrator role.

A role is a set of permissions. For example, the administrator role has permissions to perform all read, write, and delete operations. Also, the role has a scope. The scope specifies the management groups, subscriptions, or resource groups within which the role is allowed to operate.

Applying consistent permissions to resources via management groups or resource groups reduces proliferation of custom, specific, per-resource permissions. Custom resource-based permissions are often unnecessary, and can cause confusion because they do not carry their intent to new similar resources. This process can accumulate into a complex legacy configuration that is difficult to maintain or change without fear of *breaking something*, and negatively impacting both security, and solution agility.

When assigning a role to a user consider what actions the role can perform and what is the scope of those operations. Here are some considerations for role assignment:

- Use built-in roles before creating custom roles to grant the appropriate permissions to VMs and other objects. You can assign built-in roles to users, groups, service principals, and managed identities. For more information, see Azure built-in roles.

- If you need to create custom roles, grant roles with the appropriate action. Actions are categorized into operational and data actions. Start with actions that have least privilege and add more based your operational or data access needs. Provide clear guidance to your technical teams that implement permissions. For more information, see Azure custom roles.

- If you have a segmentation strategy, assign permissions with a scope. For example, if you use management group to support your strategy, set the scope to the group rather than the individual subscriptions. This will drive consistency and ensure application to future subscriptions. When assigning permissions for a segment, consider consistency while allowing flexibility to accommodate several organizational models. These models can range from a single centralized IT group to mostly independent IT and DevOps teams. For information about assigning scope, see AssignableScopes.

- You can use security groups to assign permissions. However, there are disadvantages. It can get complex because the workload needs to keep track of which security groups correspond to which application roles, for each tenant. Also, access tokens can grow significantly and Azure AD includes an "overage" claim to limit the token size. See Microsoft identity platform access tokens.

- Instead of granting permissions to specific users, assign access to Azure AD groups. In addition, build a comprehensive delegation model that includes management groups, subscription, or resource groups RBAC. For more information, see Azure role-based access control (Azure RBAC).

For information about implementing role-based authorization in an ASP.NET application, see Role-based authorization.

Learn more

- Avoid granular and custom permissions
- Delegate administration in Azure AD

Resource-based authorization

With role-based authorization, a user gets the same level of control on a resource based on the user's role. However, there might be situations where you need to define access rights per resource. For example, in a resource group, you want to allow some users to delete the resource; other users cannot. In such situations, use resource-based authorization that authorizes an action based on a particular resource. Every resource has an Owner. Owner can delete the resource. Contributors can read and update but can't delete it.

> ⓘ **Note**
>
> The *owner* and *contributor* roles for a resource are not the same as application roles.

You'll need to implement custom logic for resource-based authorization. That logic might be a mapping of resources, Azure AD object (like role, group, user), and permissions.

For information and code sample about implementing resource-based authorization in an ASP.NET application, see Resource-based authorization.

Authorization for critical accounts

There might be cases when you need to do activities that require access to important resources. Those resources might already be accessible to critical accounts such as an administrator account. Or, you might need to elevate the access permissions until the activities are complete. Both approaches can pose significant risks.

Critical accounts are those which can produce a business-critical outcome, whether cloud administrators or workload-specific privileged users. Compromise or misuse of such an account can have a detrimental-to-material effect on the business and its information systems. It's important to identify those accounts and adopt processes including close monitoring, and lifecycle management, including retirement.

Securing privileged access is a critical first step to establishing security assurances for business assets in a modern organization. The security of most or all business assets in an IT organization depends on the integrity of the privileged accounts used to administer, manage, and develop. Cyberattackers often target these accounts and other elements of privileged access to gain access to data, and systems using credential theft attacks like Pass-the-Hash, and Pass-the-Ticket.

Protecting privileged access against determined adversaries requires you to take a complete and thoughtful approach to isolate these systems from risks.

Are there any processes and tools leveraged to manage privileged activities?

Do not provide permanent access for any critical accounts and lower permissions when access is no longer required. Some strategies include:

- Just-in-time privileged access to Azure AD and Azure resources.
- Time-bound access.
- Approval-based access.
- Break glass for emergency access process to gain access.

Limit write access to production systems to service principals. No user accounts should have regular write-access.

Ensure there's a process for disabling or deleting administrative accounts that are unused.

You can use native and third-party options to elevate access permissions for at least highly privileged if not all activities. Azure AD Privileged Identity Management (Azure AD PIM) is the recommended native solution on Azure.

For more information about PIM, see What is Azure AD Privileged Identity Management?

Learn more

Establish lifecycle management for critical impact accounts

Related links

> Back to the main article: Azure identity and access management considerations

Azure Identity Management and access control security best practices

Article • 12/19/2022 • 22 minutes to read

In this article, we discuss a collection of Azure identity management and access control security best practices. These best practices are derived from our experience with Azure AD and the experiences of customers like yourself.

For each best practice, we explain:

- What the best practice is
- Why you want to enable that best practice
- What might be the result if you fail to enable the best practice
- Possible alternatives to the best practice
- How you can learn to enable the best practice

This Azure identity management and access control security best practices article is based on a consensus opinion and Azure platform capabilities and feature sets, as they exist at the time this article was written.

The intention in writing this article is to provide a general roadmap to a more robust security posture after deployment guided by our "5 steps to securing your identity infrastructure" checklist, which walks you through some of our core features and services.

Opinions and technologies change over time and this article will be updated on a regular basis to reflect those changes.

Azure identity management and access control security best practices discussed in this article include:

- Treat identity as the primary security perimeter
- Centralize identity management
- Manage connected tenants
- Enable single sign-on
- Turn on Conditional Access
- Plan for routine security improvements
- Enable password management
- Enforce multi-factor verification for users
- Use role-based access control
- Lower exposure of privileged accounts

- Control locations where resources are located
- Use Azure AD for storage authentication

Treat identity as the primary security perimeter

Many consider identity to be the primary perimeter for security. This is a shift from the traditional focus on network security. Network perimeters keep getting more porous, and that perimeter defense can't be as effective as it was before the explosion of BYOD devices and cloud applications.

Azure Active Directory (Azure AD) is the Azure solution for identity and access management. Azure AD is a multitenant, cloud-based directory and identity management service from Microsoft. It combines core directory services, application access management, and identity protection into a single solution.

The following sections list best practices for identity and access security using Azure AD.

Best practice: Center security controls and detections around user and service identities.
Detail: Use Azure AD to collocate controls and identities.

Centralize identity management

In a hybrid identity scenario we recommend that you integrate your on-premises and cloud directories. Integration enables your IT team to manage accounts from one location, regardless of where an account is created. Integration also helps your users be more productive by providing a common identity for accessing both cloud and on-premises resources.

Best practice: Establish a single Azure AD instance. Consistency and a single authoritative source will increase clarity and reduce security risks from human errors and configuration complexity. **Detail**: Designate a single Azure AD directory as the authoritative source for corporate and organizational accounts.

Best practice: Integrate your on-premises directories with Azure AD.
Detail: Use Azure AD Connect to synchronize your on-premises directory with your cloud directory.

> ⓘ **Note**
>
> There are **factors that affect the performance of Azure AD Connect**. Ensure Azure AD Connect has enough capacity to keep underperforming systems from impeding security and productivity. Large or complex organizations (organizations

provisioning more than 100,000 objects) should follow the **recommendations** to optimize their Azure AD Connect implementation.

Best practice: Don’t synchronize accounts to Azure AD that have high privileges in your existing Active Directory instance. **Detail**: Don’t change the default Azure AD Connect configuration that filters out these accounts. This configuration mitigates the risk of adversaries pivoting from cloud to on-premises assets (which could create a major incident).

Best practice: Turn on password hash synchronization.
Detail: Password hash synchronization is a feature used to synch user password hashes from an on-premises Active Directory instance to a cloud-based Azure AD instance. This sync helps to protect against leaked credentials being replayed from previous attacks.

Even if you decide to use federation with Active Directory Federation Services (AD FS) or other identity providers, you can optionally set up password hash synchronization as a backup in case your on-premises servers fail or become temporarily unavailable. This sync enables users to sign in to the service by using the same password that they use to sign in to their on-premises Active Directory instance. It also allows Identity Protection to detect compromised credentials by comparing synchronized password hashes with passwords known to be compromised, if a user has used the same email address and password on other services that aren't connected to Azure AD.

For more information, see Implement password hash synchronization with Azure AD Connect sync.

Best practice: For new application development, use Azure AD for authentication. **Detail**: Use the correct capabilities to support authentication:

- Azure AD for employees
- Azure AD B2B for guest users and external partners
- Azure AD B2C to control how customers sign up, sign in, and manage their profiles when they use your applications

Organizations that don’t integrate their on-premises identity with their cloud identity can have more overhead in managing accounts. This overhead increases the likelihood of mistakes and security breaches.

> ⓘ **Note**
>
> You need to choose which directories critical accounts will reside in and whether the admin workstation used is managed by new cloud services or existing processes. Using existing management and identity provisioning processes can

decrease some risks but can also create the risk of an attacker compromising an on-premises account and pivoting to the cloud. You might want to use a different strategy for different roles (for example, IT admins vs. business unit admins). You have two options. First option is to create Azure AD Accounts that aren't synchronized with your on-premises Active Directory instance. Join your admin workstation to Azure AD, which you can manage and patch by using Microsoft Intune. Second option is to use existing admin accounts by synchronizing to your on-premises Active Directory instance. Use existing workstations in your Active Directory domain for management and security.

Manage connected tenants

Your security organization needs visibility to assess risk and to determine whether the policies of your organization, and any regulatory requirements, are being followed. You should ensure that your security organization has visibility into all subscriptions connected to your production environment and network (via Azure ExpressRoute or site-to-site VPN). A Global Administrator in Azure AD can elevate their access to the User Access Administrator role and see all subscriptions and managed groups connected to your environment.

See elevate access to manage all Azure subscriptions and management groups to ensure that you and your security group can view all subscriptions or management groups connected to your environment. You should remove this elevated access after you've assessed risks.

Enable single sign-on

In a mobile-first, cloud-first world, you want to enable single sign-on (SSO) to devices, apps, and services from anywhere so your users can be productive wherever and whenever. When you have multiple identity solutions to manage, this becomes an administrative problem not only for IT but also for users who have to remember multiple passwords.

By using the same identity solution for all your apps and resources, you can achieve SSO. And your users can use the same set of credentials to sign in and access the resources that they need, whether the resources are located on-premises or in the cloud.

Best practice: Enable SSO.
Detail: Azure AD extends on-premises Active Directory to the cloud. Users can use their

primary work or school account for their domain-joined devices, company resources, and all of the web and SaaS applications that they need to get their jobs done. Users don't have to remember multiple sets of usernames and passwords, and their application access can be automatically provisioned (or deprovisioned) based on their organization group memberships and their status as an employee. And you can control that access for gallery apps or for your own on-premises apps that you've developed and published through the Azure AD Application Proxy.

Use SSO to enable users to access their SaaS applications based on their work or school account in Azure AD. This is applicable not only for Microsoft SaaS apps, but also other apps, such as Google Apps and Salesforce. You can configure your application to use Azure AD as a SAML-based identity provider. As a security control, Azure AD does not issue a token that allows users to sign in to the application unless they have been granted access through Azure AD. You can grant access directly, or through a group that users are a member of.

Organizations that don't create a common identity to establish SSO for their users and applications are more exposed to scenarios where users have multiple passwords. These scenarios increase the likelihood of users reusing passwords or using weak passwords.

Turn on Conditional Access

Users can access your organization's resources by using a variety of devices and apps from anywhere. As an IT admin, you want to make sure that these devices meet your standards for security and compliance. Just focusing on who can access a resource is not sufficient anymore.

To balance security and productivity, you need to think about how a resource is accessed before you can make a decision about access control. With Azure AD Conditional Access, you can address this requirement. With Conditional Access, you can make automated access control decisions based on conditions for accessing your cloud apps.

Best practice: Manage and control access to corporate resources.
Detail: Configure common Azure AD Conditional Access policies based on a group, location, and application sensitivity for SaaS apps and Azure AD–connected apps.

Best practice: Block legacy authentication protocols. **Detail**: Attackers exploit weaknesses in older protocols every day, particularly for password spray attacks. Configure Conditional Access to block legacy protocols.

Plan for routine security improvements

Security is always evolving, and it is important to build into your cloud and identity management framework a way to regularly show growth and discover new ways to secure your environment.

Identity Secure Score is a set of recommended security controls that Microsoft publishes that works to provide you a numerical score to objectively measure your security posture and help plan future security improvements. You can also view your score in comparison to those in other industries as well as your own trends over time.

Best practice: Plan routine security reviews and improvements based on best practices in your industry. **Detail**: Use the Identity Secure Score feature to rank your improvements over time.

Enable password management

If you have multiple tenants or you want to enable users to reset their own passwords, it's important that you use appropriate security policies to prevent abuse.

Best practice: Set up self-service password reset (SSPR) for your users.
Detail: Use the Azure AD self-service password reset feature.

Best practice: Monitor how or if SSPR is really being used.
Detail: Monitor the users who are registering by using the Azure AD Password Reset Registration Activity report. The reporting feature that Azure AD provides helps you answer questions by using prebuilt reports. If you're appropriately licensed, you can also create custom queries.

Best practice: Extend cloud-based password policies to your on-premises infrastructure.
Detail: Enhance password policies in your organization by performing the same checks for on-premises password changes as you do for cloud-based password changes. Install Azure AD password protection for Windows Server Active Directory agents on-premises to extend banned password lists to your existing infrastructure. Users and admins who change, set, or reset passwords on-premises are required to comply with the same password policy as cloud-only users.

Enforce multi-factor verification for users

We recommend that you require two-step verification for all of your users. This includes administrators and others in your organization who can have a significant impact if their account is compromised (for example, financial officers).

There are multiple options for requiring two-step verification. The best option for you depends on your goals, the Azure AD edition you're running, and your licensing program. See How to require two-step verification for a user to determine the best option for you. See the Azure AD and Azure AD Multi-Factor Authentication pricing pages for more information about licenses and pricing.

Following are options and benefits for enabling two-step verification:

Option 1: Enable MFA for all users and login methods with Azure AD Security Defaults
Benefit: This option enables you to easily and quickly enforce MFA for all users in your environment with a stringent policy to:

- Challenge administrative accounts and administrative logon mechanisms
- Require MFA challenge via Microsoft Authenticator for all users
- Restrict legacy authentication protocols.

This method is available to all licensing tiers but is not able to be mixed with existing Conditional Access policies. You can find more information in Azure AD Security Defaults

Option 2: Enable Multi-Factor Authentication by changing user state.
Benefit: This is the traditional method for requiring two-step verification. It works with both Azure AD Multi-Factor Authentication in the cloud and Azure AD Multi-Factor Authentication Server. Using this method requires users to perform two-step verification every time they sign in and overrides Conditional Access policies.

To determine where Multi-Factor Authentication needs to be enabled, see Which version of Azure AD MFA is right for my organization?.

Option 3: Enable Multi-Factor Authentication with Conditional Access policy. **Benefit**: This option allows you to prompt for two-step verification under specific conditions by using Conditional Access. Specific conditions can be user sign-in from different locations, untrusted devices, or applications that you consider risky. Defining specific conditions where you require two-step verification enables you to avoid constant prompting for your users, which can be an unpleasant user experience.

This is the most flexible way to enable two-step verification for your users. Enabling a Conditional Access policy works only for Azure AD Multi-Factor Authentication in the cloud and is a premium feature of Azure AD. You can find more information on this method in Deploy cloud-based Azure AD Multi-Factor Authentication.

Option 4: Enable Multi-Factor Authentication with Conditional Access policies by evaluating Risk-based Conditional Access policies.
Benefit: This option enables you to:

- Detect potential vulnerabilities that affect your organization's identities.
- Configure automated responses to detected suspicious actions that are related to your organization's identities.
- Investigate suspicious incidents and take appropriate action to resolve them.

This method uses the Azure AD Identity Protection risk evaluation to determine if two-step verification is required based on user and sign-in risk for all cloud applications. This method requires Azure Active Directory P2 licensing. You can find more information on this method in Azure Active Directory Identity Protection.

> ⓘ **Note**
>
> Option 2, enabling Multi-Factor Authentication by changing the user state, overrides Conditional Access policies. Because options 3 and 4 use Conditional Access policies, you cannot use option 2 with them.

Organizations that don't add extra layers of identity protection, such as two-step verification, are more susceptible for credential theft attack. A credential theft attack can lead to data compromise.

Use role-based access control

Access management for cloud resources is critical for any organization that uses the cloud. Azure role-based access control (Azure RBAC) helps you manage who has access to Azure resources, what they can do with those resources, and what areas they have access to.

Designating groups or individual roles responsible for specific functions in Azure helps avoid confusion that can lead to human and automation errors that create security risks. Restricting access based on the need to know and least privilege security principles is imperative for organizations that want to enforce security policies for data access.

Your security team needs visibility into your Azure resources in order to assess and remediate risk. If the security team has operational responsibilities, they need additional permissions to do their jobs.

You can use Azure RBAC to assign permissions to users, groups, and applications at a certain scope. The scope of a role assignment can be a subscription, a resource group, or a single resource.

Best practice: Segregate duties within your team and grant only the amount of access to users that they need to perform their jobs. Instead of giving everybody unrestricted permissions in your Azure subscription or resources, allow only certain actions at a particular scope. **Detail**: Use Azure built-in roles in Azure to assign privileges to users.

> ⓘ **Note**
>
> Specific permissions create unneeded complexity and confusion, accumulating into a “legacy” configuration that’s difficult to fix without fear of breaking something. Avoid resource-specific permissions. Instead, use management groups for enterprise-wide permissions and resource groups for permissions within subscriptions. Avoid user-specific permissions. Instead, assign access to groups in Azure AD.

Best practice: Grant security teams with Azure responsibilities access to see Azure resources so they can assess and remediate risk. **Detail**: Grant security teams the Azure RBAC Security Reader role. You can use the root management group or the segment management group, depending on the scope of responsibilities:

- **Root management group** for teams responsible for all enterprise resources
- **Segment management group** for teams with limited scope (commonly because of regulatory or other organizational boundaries)

Best practice: Grant the appropriate permissions to security teams that have direct operational responsibilities. **Detail**: Review the Azure built-in roles for the appropriate role assignment. If the built-in roles don't meet the specific needs of your organization, you can create Azure custom roles. As with built-in roles, you can assign custom roles to users, groups, and service principals at subscription, resource group, and resource scopes.

Best practices: Grant Microsoft Defender for Cloud access to security roles that need it. Defender for Cloud allows security teams to quickly identify and remediate risks. **Detail**: Add security teams with these needs to the Azure RBAC Security Admin role so they can view security policies, view security states, edit security policies, view alerts and recommendations, and dismiss alerts and recommendations. You can do this by using the root management group or the segment management group, depending on the scope of responsibilities.

Organizations that don’t enforce data access control by using capabilities like Azure RBAC might be giving more privileges than necessary to their users. This can lead to data compromise by allowing users to access types of data (for example, high business impact) that they shouldn’t have.

Lower exposure of privileged accounts

Securing privileged access is a critical first step to protecting business assets. Minimizing the number of people who have access to secure information or resources reduces the chance of a malicious user getting access, or an authorized user inadvertently affecting a sensitive resource.

Privileged accounts are accounts that administer and manage IT systems. Cyber attackers target these accounts to gain access to an organization's data and systems. To secure privileged access, you should isolate the accounts and systems from the risk of being exposed to a malicious user.

We recommend that you develop and follow a roadmap to secure privileged access against cyber attackers. For information about creating a detailed roadmap to secure identities and access that are managed or reported in Azure AD, Microsoft Azure, Microsoft 365, and other cloud services, review Securing privileged access for hybrid and cloud deployments in Azure AD.

The following summarizes the best practices found in Securing privileged access for hybrid and cloud deployments in Azure AD:

Best practice: Manage, control, and monitor access to privileged accounts.
Detail: Turn on Azure AD Privileged Identity Management. After you turn on Privileged Identity Management, you'll receive notification email messages for privileged access role changes. These notifications provide early warning when additional users are added to highly privileged roles in your directory.

Best practice: Ensure all critical admin accounts are managed Azure AD accounts. **Detail**: Remove any consumer accounts from critical admin roles (for example, Microsoft accounts like hotmail.com, live.com, and outlook.com).

Best practice: Ensure all critical admin roles have a separate account for administrative tasks in order to avoid phishing and other attacks to compromise administrative privileges. **Detail**: Create a separate admin account that's assigned the privileges needed to perform the administrative tasks. Block the use of these administrative accounts for daily productivity tools like Microsoft 365 email or arbitrary web browsing.

Best practice: Identify and categorize accounts that are in highly privileged roles.
Detail: After turning on Azure AD Privileged Identity Management, view the users who are in the global administrator, privileged role administrator, and other highly privileged roles. Remove any accounts that are no longer needed in those roles, and categorize the remaining accounts that are assigned to admin roles:

- Individually assigned to administrative users, and can be used for non-administrative purposes (for example, personal email)
- Individually assigned to administrative users and designated for administrative purposes only
- Shared across multiple users
- For emergency access scenarios
- For automated scripts
- For external users

Best practice: Implement “just in time” (JIT) access to further lower the exposure time of privileges and increase your visibility into the use of privileged accounts.
Detail: Azure AD Privileged Identity Management lets you:

- Limit users to only taking on their privileges JIT.
- Assign roles for a shortened duration with confidence that the privileges are revoked automatically.

Best practice: Define at least two emergency access accounts.
Detail: Emergency access accounts help organizations restrict privileged access in an existing Azure Active Directory environment. These accounts are highly privileged and are not assigned to specific individuals. Emergency access accounts are limited to scenarios where normal administrative accounts can’t be used. Organizations must limit the emergency account's usage to only the necessary amount of time.

Evaluate the accounts that are assigned or eligible for the global admin role. If you don’t see any cloud-only accounts by using the `*.onmicrosoft.com` domain (intended for emergency access), create them. For more information, see Managing emergency access administrative accounts in Azure AD.

Best practice: Have a “break glass" process in place in case of an emergency. **Detail**: Follow the steps in Securing privileged access for hybrid and cloud deployments in Azure AD.

Best practice: Require all critical admin accounts to be password-less (preferred), or require Multi-Factor Authentication. **Detail**: Use the Microsoft Authenticator app to sign in to any Azure AD account without using a password. Like Windows Hello for Business, the Microsoft Authenticator uses key-based authentication to enable a user credential that’s tied to a device and uses biometric authentication or a PIN.

Require Azure AD Multi-Factor Authentication at sign-in for all individual users who are permanently assigned to one or more of the Azure AD admin roles: Global Administrator, Privileged Role Administrator, Exchange Online Administrator, and

SharePoint Online Administrator. Enable Multi-Factor Authentication for your admin accounts and ensure that admin account users have registered.

Best practice: For critical admin accounts, have an admin workstation where production tasks aren't allowed (for example, browsing and email). This will protect your admin accounts from attack vectors that use browsing and email and significantly lower your risk of a major incident. **Detail**: Use an admin workstation. Choose a level of workstation security:

- Highly secure productivity devices provide advanced security for browsing and other productivity tasks.
- Privileged Access Workstations (PAWs) provide a dedicated operating system that's protected from internet attacks and threat vectors for sensitive tasks.

Best practice: Deprovision admin accounts when employees leave your organization. **Detail**: Have a process in place that disables or deletes admin accounts when employees leave your organization.

Best practice: Regularly test admin accounts by using current attack techniques. **Detail**: Use Microsoft 365 Attack Simulator or a third-party offering to run realistic attack scenarios in your organization. This can help you find vulnerable users before a real attack occurs.

Best practice: Take steps to mitigate the most frequently used attacked techniques. **Detail**: Identify Microsoft accounts in administrative roles that need to be switched to work or school accounts

Ensure separate user accounts and mail forwarding for global administrator accounts

Ensure that the passwords of administrative accounts have recently changed

Turn on password hash synchronization

Require Multi-Factor Authentication for users in all privileged roles as well as exposed users

Obtain your Microsoft 365 Secure Score (if using Microsoft 365)

Review the Microsoft 365 security guidance (if using Microsoft 365)

Configure Microsoft 365 Activity Monitoring (if using Microsoft 365)

Establish incident/emergency response plan owners

Secure on-premises privileged administrative accounts

If you don’t secure privileged access, you might find that you have too many users in highly privileged roles and are more vulnerable to attacks. Malicious actors, including cyber attackers, often target admin accounts and other elements of privileged access to gain access to sensitive data and systems by using credential theft.

Control locations where resources are created

Enabling cloud operators to perform tasks while preventing them from breaking conventions that are needed to manage your organization's resources is very important. Organizations that want to control the locations where resources are created should hard code these locations.

You can use Azure Resource Manager to create security policies whose definitions describe the actions or resources that are specifically denied. You assign those policy definitions at the desired scope, such as the subscription, the resource group, or an individual resource.

> **Note**
>
> Security policies are not the same as Azure RBAC. They actually use Azure RBAC to authorize users to create those resources.

Organizations that are not controlling how resources are created are more susceptible to users who might abuse the service by creating more resources than they need. Hardening the resource creation process is an important step to securing a multitenant scenario.

Actively monitor for suspicious activities

An active identity monitoring system can quickly detect suspicious behavior and trigger an alert for further investigation. The following table lists Azure AD capabilities that can help organizations monitor their identities:

Best practice: Have a method to identify:

- Attempts to sign in without being traced.
- Brute force attacks against a particular account.
- Attempts to sign in from multiple locations.
- Sign-ins from infected devices.
- Suspicious IP addresses.

Detail: Use Azure AD Premium anomaly reports. Have processes and procedures in place for IT admins to run these reports on a daily basis or on demand (usually in an incident response scenario).

Best practice: Have an active monitoring system that notifies you of risks and can adjust risk level (high, medium, or low) to your business requirements.
Detail: Use Azure AD Identity Protection, which flags the current risks on its own dashboard and sends daily summary notifications via email. To help protect your organization's identities, you can configure risk-based policies that automatically respond to detected issues when a specified risk level is reached.

Organizations that don't actively monitor their identity systems are at risk of having user credentials compromised. Without knowledge that suspicious activities are taking place through these credentials, organizations can't mitigate this type of threat.

Use Azure AD for storage authentication

Azure Storage supports authentication and authorization with Azure AD for Blob storage and Queue storage. With Azure AD authentication, you can use the Azure role-based access control to grant specific permissions to users, groups, and applications down to the scope of an individual blob container or queue.

We recommend that you use Azure AD for authenticating access to storage.

Next step

See Azure security best practices and patterns for more security best practices to use when you're designing, deploying, and managing your cloud solutions by using Azure.

Additional resources

Documentation

Azure AD security for AWS - Azure Architecture Center

Learn how Azure Active Directory can help secure and protect Amazon Web Services (AWS) identity management and account access. Discover Microsoft security solutions.

Best practices to secure with Azure Active Directory - Microsoft Entra

Best practices we recommend you follow to secure your isolated environments in Azure Active Directory.

Secure access practices for administrators in Azure AD - Microsoft Entra

Ensure that your organization's administrative access and administrator accounts are secure. For system architects and IT pros who configure Azure AD, Azure, and Microsoft Online Services.

Azure security features that help with identity management

Learn about the core Azure security features that help with identity management. See information about topics like single sign-on and reverse proxy.

Azure security baseline for Azure Active Directory

The Azure Active Directory security baseline provides procedural guidance and resources for implementing the security recommendations specified in the Azure Security Benchmark.

Architecture overview - Azure Active Directory - Microsoft Entra

Learn what an Azure Active Directory tenant is and how to manage Azure using Azure Active Directory.

Secure your Azure AD identity infrastructure - Azure Active Directory

This document outlines a list of important actions administrators should implement to help them secure their organization using Azure AD capabilities

Securing identity with Zero Trust

Identities, representing people, services, or IoT devices, are the common dominator across today's many networks, endpoints, and applications. In the Zero Trust security model, they function as a powerful, flexible, and granular way to control access to data.

Show 5 more

Training

Learning path

SC-300: Implement an Authentication and Access Management solution - Training

Implement and administer your access management using Azure AD. Use MFA, Conditional Access, and identity protection to manager your identity solution. Aligned to SC-300 exam.

Certification

Microsoft Certified: Identity and Access Administrator Associate - Certifications

The Microsoft identity and access administrator designs, implements, and operates an organization's identity and access management systems by using Microsoft Azure Active Directory (Azure AD), part of Microsoft Entra. They configure and manage authentication and authorization of identities for...

Network security

Article • 04/21/2023

Protect assets by placing controls on network traffic originating in Azure, between on-premises and Azure hosted resources, and traffic to and from Azure. If security measures aren't in place, attackers can gain access, for instance, by scanning across public IP ranges. Proper network security controls can provide defense-in-depth elements that help detect, contain, and stop attackers who gain entry into your cloud deployments.

> **Note**
>
> Network security, segmentation, and connectivity could be defined as part of the workload architecture. More commonly, networking is often addressed at an organizational level by central IT, cloud center of excellence, or a cloud platform team. For any networking configuration defined outside of the scope of your workload architecture, reference Cloud Adoption Framework's **Azure landing zones**. **Network topology and connectivity** outlines the best practice recommendations and considerations for centralized networking and network security.

Checklist

How have you secured the network of your workload?

- ✓ Segment your network footprint and create secure communication paths between segments. Align the network segmentation with overall enterprise segmentation strategy.
- ✓ Design security controls that identify and allow or deny traffic, access requests, and application communication between segments.
- ✓ Protect all public endpoints with Azure Front Door, Application Gateway, Azure Firewall, and Azure DDoS Protection.
- ✓ Mitigate DDoS attacks with Azure DDoS Protection for critical workloads.
- ✓ Keep virtual machines private and secure when connecting to the internet with Azure NAT Gateway.
- ✓ Control network traffic between subnets (east-west) and application tiers (north-south).
- ✓ Protect from data exfiltration attacks through a defense-in-depth approach with controls at each layer.

Azure security benchmark

The Azure Security Benchmark includes a collection of high-impact security recommendations you can use to help secure the services you use in Azure:

> The questions in this section are aligned to the Azure Security Benchmarks Network Security.

Azure services

- Azure Virtual Network
- Azure Firewall
- Azure NAT Gateway
- Azure ExpressRoute
- Azure Private Link
- Azure DDoS Protection

Reference architecture

Here are some reference architectures related to network security:

- Hub-spoke network topology in Azure
- Deploy highly available NVAs
- Baseline architecture for an Azure Kubernetes Service (AKS) cluster

Next steps

We recommend applying as many as of the best practices as early as possible, and then working to retrofit any gaps over time as you mature your security program.

Data protection

Related links

Combine network controls with application, identity, and other technical control types. This approach is effective in preventing, detecting, and responding to threats outside the networks you control. For more information, see these articles:

- Applications and services

- Azure identity and access management considerations
- Data protection considerations

Ensure that resource grouping and administrative privileges align to the segmentation model. For more information, see Administrative account security.

> Go back to the main article: Overview of the security pillar

Implement network segmentation patterns on Azure

Article • 11/30/2022

A unified enterprise segmentation strategy guides technical teams to consistently segment access using networking, applications, identity, and any other access controls. Create segmentation in your network footprint by defining perimeters. The main reasons for segmentation are:

- The ability to group related assets that are a part of (or support) workload operations.
- Isolation of resources.
- Governance policies set by the organization.

Assume compromise is the recommended cybersecurity mindset and the ability to contain an attacker is vital in protecting information systems. Model an attacker able to achieve a foothold at various points within the workload and establish controls to mitigate further expansion.

Network controls can secure interactions between perimeters. This approach can strengthen the security posture and contain risks in a breach because the controls can detect, contain, and stop attackers from gaining access to an entire workload.

Containment of attack vectors within an environment is critical. However, to be effective in cloud environments, traditional approaches may prove inadequate and security organizations may need to evolve their methods.

Traditional segmentation approaches typically fail to achieve their goals as they have not been developed in a method to align with business use cases and application workloads. Often this results in overwhelming complexity requiring broad firewall exceptions.

An evolving emerging best practice recommendation is to adopt a Zero Trust strategy based on user, device, and application identities. In contrast to network access controls that are based on elements such as source and destination IP address, protocols, and port numbers, Zero Trust enforces and validates access control at *access time*. This avoids the need to play a prediction game for an entire deployment, network, or subnet — only the destination resource needs to provide the necessary access controls.

- Azure Network Security Groups can be used for basic layer 3 and 4 access controls between Azure Virtual Networks, their subnets, and the internet.

- Azure Web Application Firewall and the Azure Firewall can be used for more advanced network access controls that require application layer support.
- Local Admin Password Solution (LAPS) or a third-party Privileged Access Management can set strong local admin passwords and just-in-time access to them.

How does the organization implement network segmentation?

This article highlights some Azure networking features that create segments and restrict access to individual services.

> ⓘ **Important**
>
> Align your network segmentation strategy with the enterprise segmentation model. This will reduce confusion and challenges with different technical teams (networking, identity, applications, and so on). Each team should not develop their own segmentation and delegation models that don't align with each other.

Key points

- Create software-defined perimeters in your networking footprint and secure communication paths between them.
- Establish a complete zero trust segmentation strategy.
- Align technical teams in the enterprise on micro segmentation strategies for legacy applications.
- Azure Virtual Networks (VNets) are created in private address spaces. By default, no traffic is allowed between any two VNets. Open paths only when it's really needed.
- Use Network Security Groups (NSG) to secure communication between resources within a VNet.
- Use Application Security Groups (ASGs) to define traffic rules for the underlying VMs that run the workload.
- Use Azure Firewall to filter traffic flowing between cloud resources, the internet, and on-premise.
- Place resources in a single VNet, if you don't need to operate in multiple regions.
- If you need to be in multiple regions, have multiple VNets that are connected through peering.
- For advanced configurations, use a hub-spoke topology. A VNet is designated as a hub in a given region for all the other VNets as spokes in that region.

What is segmentation?

You can create software-defined perimeters in your networking footprint by using the various Azure services and features. When a workload (or parts of a given workload) is placed into separate segments, you can control traffic from/to those segments to secure communication paths. If a segment is compromised, you will be able to better contain the impact and prevent it from laterally spreading through the rest of your network. This strategy aligns with the key principle of Zero Trust model published by Microsoft that aims to bring world class security thinking to your organization.

Suggested actions

Create a risk containment strategy that blends proven approaches including:

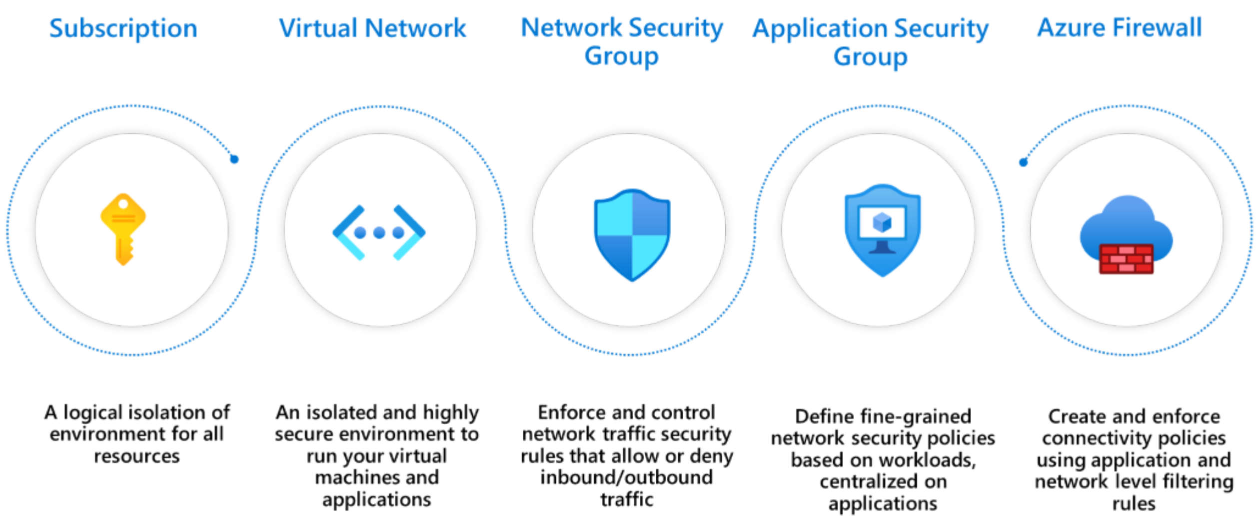
- Existing network security controls and practices
- Native security controls available in Azure
- Zero trust approaches

Learn more

For information about creating a segmentation strategy, see Enterprise segmentation strategy.

Azure features for segmentation

When you operate on Azure, you have many segmentation options.



1. Subscription: A high-level construct, which provides platform powered separation between entities. It's intended to carve out boundaries between large

organizations within a company and communication between resources in different subscriptions needs to be explicitly provisioned.

2. Virtual Network (VNets): Created within a subscription in private address spaces. They provide network level containment of resources with no traffic allowed by default between any two virtual networks. Like subscriptions, any communication between virtual networks needs to be explicitly provisioned.

3. Network Security Groups (NSG): An access control mechanisms for controlling traffic between resources within a virtual network and also with external networks, such as the internet, other virtual networks. NSGs can take your segmentation strategy to a granular level by creating perimeters for a subnet, a VM, or a group of VMs. For information about possible operations with subnets in Azure, see Subnets (Azure Virtual Networks).

4. Application Security Groups (ASGs): Similar to NSGs but are referenced with an application context. It allows you to group a set of VMs under an application tag and define traffic rules that are then applied to each of the underlying VMs.

5. Azure Firewall: A cloud native stateful Firewall as a service, which can be deployed in your VNet or in Azure Virtual WAN hub deployments for filtering traffic flowing between cloud resources, the internet, and on-premise. You create rules or policies (using Azure Firewall or Azure Firewall Manager) specifying allow/deny traffic using layer 3 to layer 7 controls. You can also filter traffic going to the internet using both Azure Firewall and third parties by directing some or all traffic through third-party security providers for advanced filtering & user protection.

Segmentation patterns

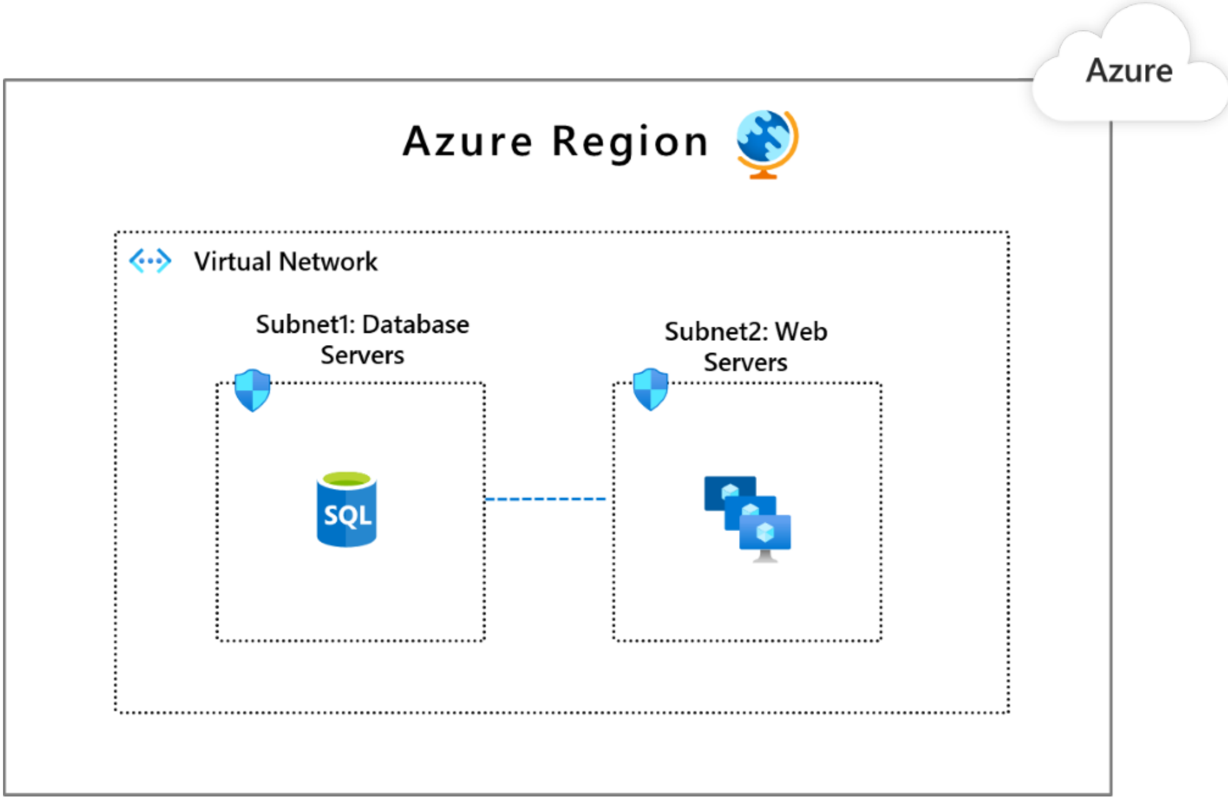
Here are some common patterns for segmenting a workload in Azure from a networking perspective. Each pattern provides a different type of isolation and connectivity. Choose a pattern based on your organization's needs.

Pattern 1: Single VNet

All the components of the workload reside in a single VNet. This pattern is appropriate you are operating in a single region because a VNet cannot span multiple regions.

Common ways for securing segments, such as subnets or application groups, are by using NSGs and ASGs. You can also use a Network Virtualized Appliance (NVAs) from Azure Marketplace or Azure Firewall to enforce and secure this segmentation.

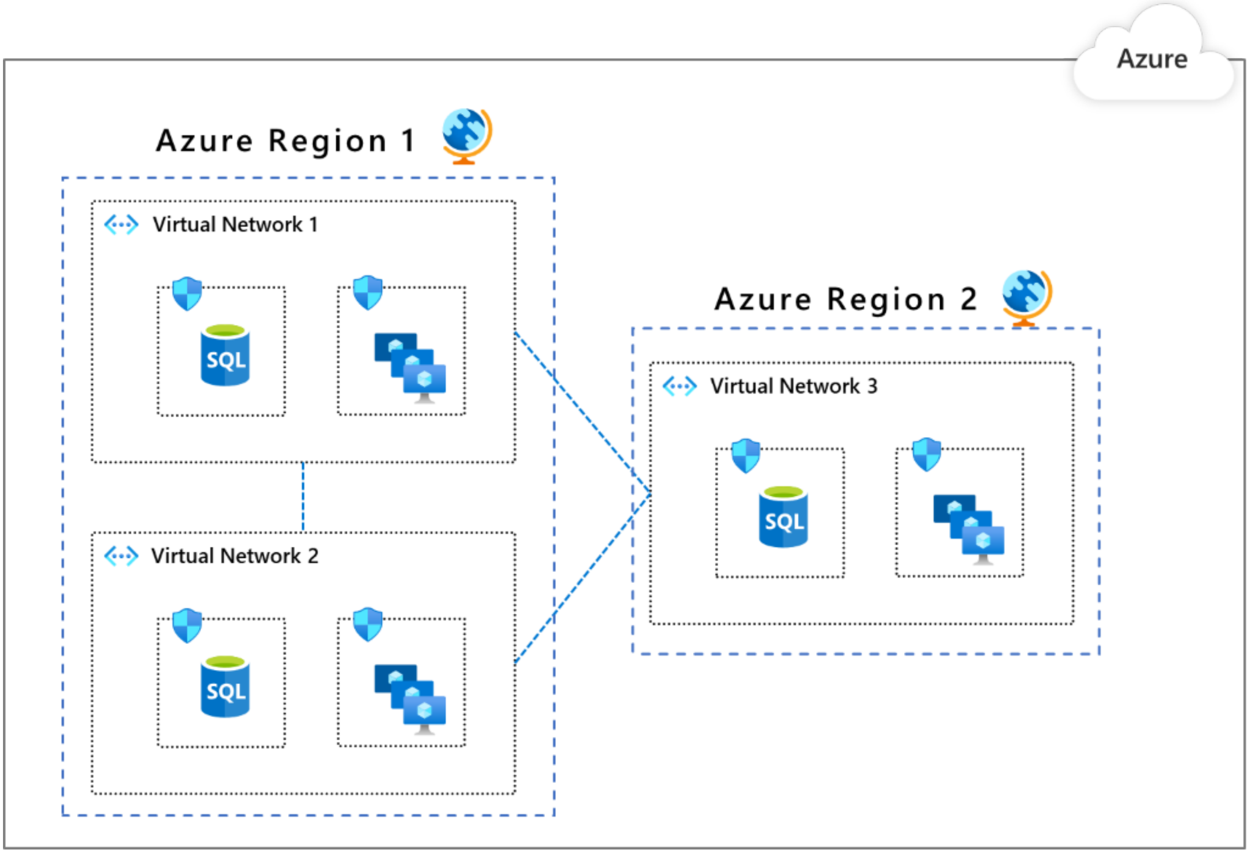
In this image, Subnet1 has the database workload. Subnet2 has the web workloads. You can configure NSGs that allow Subnet1 to only communicate with Subnet2 and Subnet2 can only communicate with the internet.



Consider a use case where you have multiple workloads that are placed in separate subnets. You can place controls that will allow one workload to communicate to the backend of another workload.

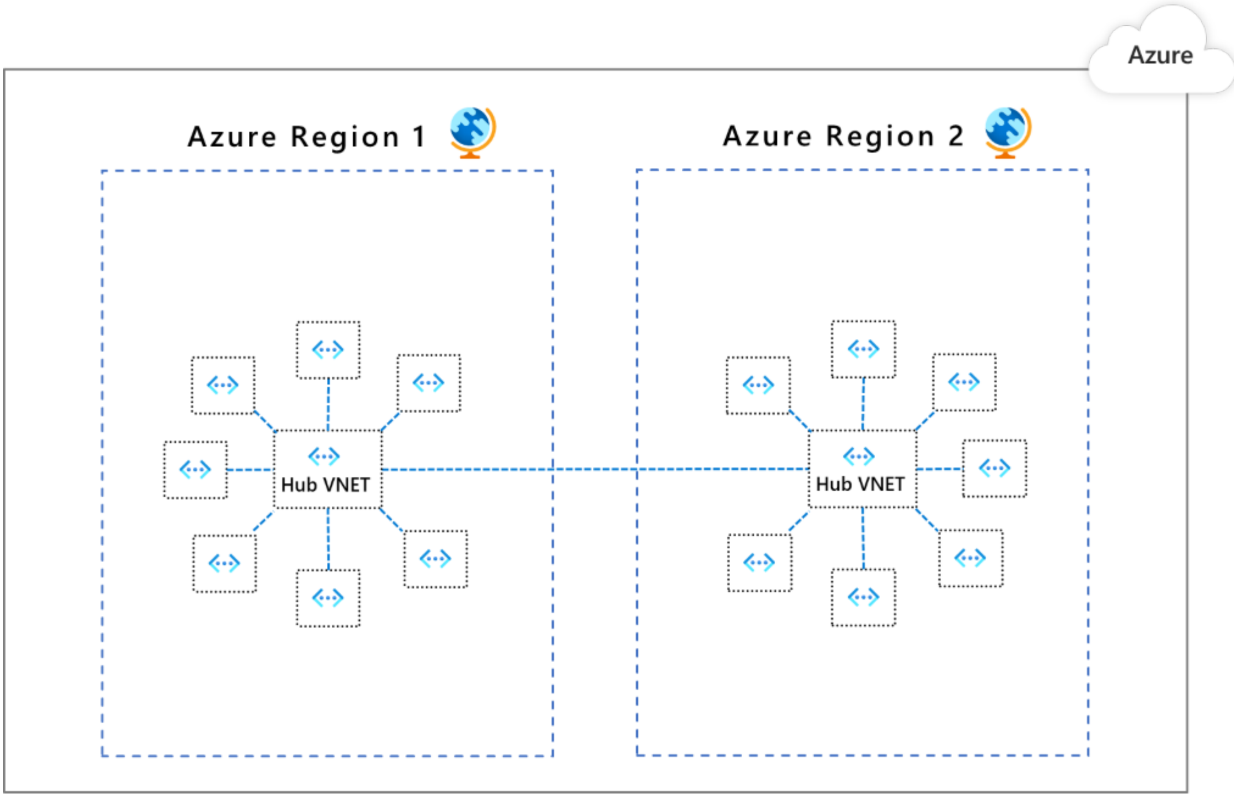
Pattern 2: Multiple VNets that communicate through with peering

The resources are spread or replicated in multiple VNets. The VNets can communicate through peering. This pattern is appropriate when you need to group applications into separate VNets. Or, you need multiple Azure regions. One benefit is the built-in segmentation because you have to explicitly peer one VNet to another. Virtual network peering is not transitive. You can further segment within a VNet by using NSGs and ASGs as shown in pattern 1.



Pattern 3: Multiple VNets in a hub and spoke model

A VNet is designated as a *hub* in a given region for all the other VNets as *spokes* in that region. The hub and its spokes are connected through peering. All traffic passes through the hub that can act as a gateway to other hubs in different regions. In this pattern, the security controls are set up at the hubs so that they get to segment and govern the traffic in between other VNets in a scalable way. One benefit of this pattern is, as your network topology grows, the security posture overhead does not grow (except when you expand to new regions).



The recommended native option is Azure Firewall. This option works across both VNets and subscriptions to govern traffic flows using layer 3 to layer 7 controls. You can define your communication rules and apply them consistently. Here are some examples:

- VNet 1 cannot communicate with VNet 2, but it can communicate VNet 3.
- VNet 1 cannot access public internet except for *.github.com.

With Azure Firewall Manager preview, you can centrally manage policies across multiple Azure Firewalls and enable DevOps teams to further customize local policies.

> **Tip**
>
> Here are some resources that illustrate provisioning of resources in a hub and spoke topology:
>
> **GitHub: Hub and Spoke Topology Sandbox** .
>
> The design considerations are described in **Hub-spoke network topology in Azure**.

Pattern comparison

Considerations	Pattern 1	Pattern 2	Pattern 3

Considerations	Pattern 1	Pattern 2	Pattern 3
Connectivity/routing: how each segment communicates to each other	System routing provides default connectivity to any workload in any subnet.	Same as a pattern 1.	No default connectivity between spoke networks. A layer 3 router, such as the Azure Firewall, in the hub is required to enable connectivity.
Network level traffic filtering	Traffic is allowed by default. Use NSG, ASG to filter traffic.	Same as a pattern 1.	Traffic between spoke virtual networks is denied by default. Open selected paths to allow traffic through Azure Firewall configuration.
Centralized logging	NSG, ASG logs for the virtual network.	Aggregate NSG, ASG logs across all virtual networks.	Azure Firewall logs all accepted/denied traffic sent through the hub. View the logs in Azure Monitor.
Unintended open public endpoints	DevOps can accidentally open a public endpoint through incorrect NSG, ASG rules.	Same as a pattern 1.	Accidentally opened public endpoint in a spoke will not enable access because the return packet will get dropped through stateful firewall (asymmetric routing).
Application level protection	NSG or ASG provides network layer support only.	Same as a pattern 1.	Azure Firewall supports FQDN filtering for HTTP/S and MSSQL for outbound traffic and across virtual networks.

Next step

Secure network connectivity

Related links

- For information about setting up peering, reference Virtual network peering.
- For best practices about using Azure Firewall in various configurations, reference Azure Firewall Architecture Guide.
- For information about different access policies and control flow within a VNet, reference Azure Virtual Network Subnet

> Back to the main article: Network security

Azure services for securing network connectivity

Article • 11/30/2022

It's often the case that the workload and the supporting components of a cloud architecture will need to access external assets. These assets can be on-premises, devices outside the main virtual network, or other Azure resources. Those connections can be over the internet or networks within the organization.

Key points

- Protect non-publicly accessible services with network restrictions and IP firewall.
- Use Network Security Groups (NSGs) or Azure Firewall to protect and control traffic within the VNet.
- Use Service Endpoints or Private Link for accessing Azure PaaS services.
- Use Azure Firewall to protect against data exfiltration attacks.
- Restrict access to backend services to a minimal set of public IP addresses.
- Use Azure controls over third-party solutions for basic security needs. These controls are native to the platform and are easy to configure and scale.
- Define access policies based on the type of workload and control flow between the different application tiers.

Connectivity between network segments

When designing a workload, you'll typically start by provisioning an Azure Virtual Network (VNet) in a private address space which has the workload. No traffic is allowed by default between any two virtual networks. If there's a need, define the communication paths explicitly. One way of connecting VNets is through Virtual network peering.

A key aspect of protecting VMs in a VNet is to control the flow of network traffic. The network interfaces on the VMs allow them to communicate with other VMs, the internet, and on-premises networks. To control traffic on VMs within a VNet (and subnet), use Application Security Groups (ASGs). ASGs allow you to group a set of VMs under an application tag and define traffic rules. Those rules are then applied to each of the underlying VMs.

A VNet is segmented into subnets based on business requirements. Ensure that proper network security controls are configured to allow or deny inbound network traffic to, or outbound network traffic from, within larger network space.

By default VMs are provisioned with private IP addresses. This allows you to take advantage of the Azure IP address to determine incoming traffic, how and where it's translated on to the virtual network.

A good Azure IP addressing schema provides flexibility, room for growth, and integration with on-premises networks. The schema ensures that communication works for deployed resources, minimizes public exposure of systems, and gives the organization flexibility in its network. If not properly designed, systems might not be able to communicate, and additional work will be required to remediate.

How do you isolate and protect traffic within the workload VNet?

To secure communication within a VNet, set rules that inspect traffic. Then, *allow* or *deny* traffic to, or from specific sources, and route them to the specified destinations.

> Review the rule set and confirm that the required services are not unintentionally blocked.

For traffic between subnets (also referred to as east-west traffic), it's recommended to use Network Security Groups (NSG). NSGs allow you to define rules that check the source and destination address, protocol and port of Inbound and Outbound traffic. The address can be a single IP address, multiple IP addresses, an Azure service tag or an entire subnet.

If NSGs are being used to isolate and protect the application, the rule set should be reviewed to confirm that required services are not unintentionally blocked, or more permissive access than expected is allowed.

For advanced networking controls, use Azure Firewall. It can be used to perform deep packet inspection on both east-west and north-south traffic. Firewalls rules can be defined as policies and centrally managed. An alternative solution is to use network virtual appliances (NVAs) that check inbound (ingress) and outbound (egress) traffic and filters based on rules.

How do you route network traffic through NVAs for security boundary policy enforcement, auditing, and inspection?

Use User Defined Routes (UDR) to control the next hop for traffic between Azure, on-premises, and internet resources. The routes can be applied to virtual appliance, virtual network gateway, virtual network, or internet.

For example, you need to inspect all ingress traffic from a public load balancer. One way is to host an NVA in a subnet that allows traffic only if certain criteria is met. That traffic is sent to the subnet that hosts an internal load balancer that routes that traffic to the backend services.

You can also use NVAs for egress traffic. For instance, all workload traffic is routed by using UDR to another subnet. That subnet has an internal load balancer that distributes requests to the NVA (or a set of NVAs). These NVAs direct traffic to the internet using their individual public IP addresses.

> **Tip**
>
> Here are the resources for the preceding example:
>
> **GitHub: Automated failover for network virtual appliances**.
>
> The design considerations are described in **Deploy highly available NVAs**.

Azure Firewall can serve as an NVA. Azure supports third-party network device providers. They're available in Azure Marketplace.

How do you get insights about ingoing and outgoing traffic of this workload?

As a general rule, configure and collect network traffic logs. If you use NSGs, capture and analyze NSG flow logs to monitor performance and security. The NSG flow logs enable Traffic Analytics to gain insights into internal and external traffic flows of the application.

For information about defining network perimeters, see Network segmentation.

Can the VNet and subnet handle growth?

Typically, you'll add more network resources as the design matures. Most organizations end up adding more resources to networks than initially planned. Refactoring to accommodate the extra resources is a labor-intensive process. There is limited security value in creating a very large number of small subnets and then trying to map network access controls (such as security groups) to each of them.

Plan your subnets based on roles and functions that use the same protocols. That way, you can add resources to the subnet without making changes to security groups that enforce network level access controls.

Don't use all open rules that allow inbound and outbound traffic to and from 0.0.0.0-255.255.255.255. Use a least-privilege approach and only allow relevant protocols. It will reduce your overall network attack surface on the subnet. All open rules provide a false sense of security because such a rule enforces no security.

The exception is when you want to use security groups only for network logging purposes.

Design virtual networks and subnets for growth. We recommend planning subnets based on common roles and functions that use common protocols for those roles and functions. This allows you to add resources to the subnet without making changes to security groups that enforce network level access controls.

Suggested actions

Use NSG or consider using Azure Firewall to protect and control traffic within the VNET.

Learn more

- Azure firewall documentation
- Design virtual network subnet security
- Design an IP addressing schema for your Azure deployment
- Network security groups

Internet edge traffic

As you design the workload, consider security for internet traffic. Does the workload or parts of it need to be accessible from public IP addresses? What level of access should be given to prevent unauthorized access?

Internet edge traffic (also called *North-South traffic*) represents network connectivity between resources used by the workload and the internet. An internet edge strategy should be designed to mitigate as many attacks from the internet to detect or block threats. There are two primary choices that provide security controls and monitoring:

- Azure solutions such as Azure Firewall and Web Application Firewall (WAF).

Azure provides networking solutions to restrict access to individual services. Use multiple levels of security, such as combination of IP filtering, firewall rules to prevent application services from being accessed by unauthorized actors.

- Network virtual appliances (NVAs). You can use Azure Firewall or third-party solutions available in Azure Marketplace.

Azure security features are sufficient for common attacks, easy to configure, and scale. Third-party solutions often have advanced features but they can be hard to configure if they don't integrate well with fabric controllers. From a cost perspective, Azure options tend to be cheaper than partner solutions.

Information revealing the application platform, such as HTTP banners containing framework information (`X-Powered-By`, `X-ASPNET-VERSION`), are commonly used by malicious actors when mapping attack vectors of the application.

HTTP headers, error messages, and website footers should not contain information about the application platform. Azure CDN can be used to separate the hosting platform from end users. Azure API Management offers transformation policies that allow you to modify HTTP headers and remove sensitive information.

Suggested action

Consider using CDN for the workload to limit platform detail exposure to attackers.

Learn more

Azure CDN documentation

Communication with backend services

Most workloads are composed of multiple tiers where several services can serve each tier. Common examples of tiers are web front ends, business processes, reporting and analysis, backend infrastructure, and so on.

Application resources allowing multiple methods to publish app content (such as FTP, Web Deploy) should have the unused endpoints disabled. For Azure Web Apps, SCM is the recommended endpoint and it can be protected separately with network restrictions for sensitive scenarios.

Public access to any workload should be judiciously approved and planned, as public entry points represent a key possible vector of compromise. When allowing access from public IPs to any back-end service, limiting the range of allowed IPs can significantly reduce the attack surface of that service. For example, if using Azure Front Door, you can limit backend tiers to allow Front Door IPs only; or if a partner uses your API, limit access to only their nominated public IP(s).

How do you configure traffic flow between multiple application tiers?

Use Azure Virtual Network Subnet to allocate separate address spaces for different elements or tiers within the workload. Then, define different access policies to control traffic flows between those tiers and restrict access. You can implement those restrictions through IP filtering or firewall rules.

Do you need to restrict access to the backend infrastructure?

Restrict access to backend services to a minimal set of public IP addresses with App Services IP restrictions or Azure Front Door.

Web applications typically have one public entry point and don't expose subsequent APIs and database servers over the internet. Expose only a minimal set of public IP addresses based on need *and* only those who really need it. For example, when using gateway services, such as Azure Front Door, it's possible to restrict access only to a set of Front Door IP addresses and lock down the infrastructure completely.

Suggested action

- Restrict and protect application publishing methods.

Learn more

- Set up Azure App Service access restrictions
- Azure Front Door documentation
- Deploy your app to Azure App Service using FTP/S

Connection with Azure PaaS services

The workload will often need to communicate with other Azure services. For example, it might need to get secrets from Azure Key Vault. Avoid making connections over the public internet.

Does the workload use secure ways to access Azure PaaS services?

Common approaches for accessing PaaS services are Service Endpoints or Private Links. Both approaches restrict access to PaaS endpoints only from authorized virtual networks, effectively mitigating data intrusion risks and associated impact to application availability.

With Service Endpoints, the communication path is secure because you can reach the PaaS endpoint without needing a public IP address on the VNet. Most PaaS services support communication through service endpoints. For a list of generally available services, see Virtual Network service endpoints.

Another mechanism is through Azure Private Link. Private Endpoint uses a private IP address from your VNet, effectively bringing the service into your VNet. For details, see What is Azure Private Link?.

Service Endpoints provide service level access to a PaaS service, whereas Private Link provides direct access to a specific PaaS resource to mitigate data exfiltration risks, such as malicious admin access. Private Link is a paid service and has meters for inbound and outbound data processed. Private Endpoints are also charged.

How do you control outgoing traffic of Azure PaaS services where Private Link isn't available?

Use NVAs and Azure Firewall (for supported protocols) as a reverse proxy to restrict access to only authorized PaaS services for services where Private Link isn't supported. Use Azure Firewall to protect against data exfiltration concerns.

On-premises to cloud connectivity

In a hybrid architecture, the workload runs partly on-premises and partly in Azure. Have security controls that check traffic entering Azure virtual network from on-premises data center.

How do you establish cross premises connectivity?

Use Azure ExpressRoute to set up cross premises connectivity to on-premises networks. This service uses a private, dedicated connection through a third-party connectivity provider. The private connection extends your on-premises network into Azure. This way, you can reduce the risk of potential of access to company's information assets on-premises.

How do you access VMs?

Use Azure Bastion to log into your VMs and avoid public internet exposure using SSH and RDP with private IP addresses only. You can also disable RDP/SSH access to VMs and use VPN, ExpressRoute to access these virtual machines for remote management.

Do the cloud or on-premises VMs have direct internet connectivity for users that may perform interactive logins?

Attackers constantly scan public cloud IP ranges for open management ports and attempt low-cost attacks such as common passwords and known unpatched vulnerabilities. Develop processes and procedures to prevent direct internet access of VMs with logging and monitoring to enforce policies.

How is internet traffic routed?

Decide how to route internet traffic. You can use on-premises security devices (also called *forced tunneling*) or allow connectivity through cloud-based network security devices.

For production enterprise, allow cloud resources to start and respond to internet request directly through cloud network security devices defined by your internet edge strategy. This approach fits the Nth datacenter paradigm, that is Azure datacenters are a part of your enterprise. It scales better for an enterprise deployment because it removes hops that add load, latency, and cost.

Another option is to force tunnel all outbound internet traffic from on-premises through site-to-site VPN. Or, use a cross-premise WAN link. Network security teams have greater security and visibility to internet traffic. Even when your resources in the cloud try to respond to incoming requests from the internet, the responses are force tunneled. This option fits a datacenter expansion use case and can work well for a quick proof of concept, but scales poorly because of the increased traffic load, latency, and cost. For those reasons, we recommend that you avoid forced tunneling.

Next step

Secure endpoints

Related links

For information about controlling next hop for traffic, see Azure Virtual Network User Defined Routes (UDR).

For information about web application firewalls, see Application Gateway WAF.

For information about Network Appliances from Azure Marketplace, see Network Appliances.

For information about cross premises connectivity, see Azure site-to-site VPN or ExpressRoute.

For information about using VPN/ExpressRoute to access these virtual machines for remote management, see Disable RDP/SSH access to Azure Virtual Machines.

> Go back to the main article: Network security

Best practices for endpoint security on Azure

Article • 11/30/2022

An *endpoint* is an address exposed by a web application so that external entities can communicate with it. A malicious or an inadvertent interaction with the endpoint can compromise the security of the application and even the entire system. One way to protect the endpoint is by placing filter controls on the network traffic that it receives, such as defining rule sets. A defense-in-depth approach can further mitigate risks. Include supplemental controls that protect the endpoint if the primary traffic controls fail.

This article describes way in which you can protect web applications with Azure services and features. For product documentation, see Related links.

Key points

- Protect all public endpoints with Azure Front Door, Application Gateway, Azure Firewall, Azure DDoS Protection.
- Use web application firewall (WAF) to protect web workloads.
- Protect workload publishing methods and restrict ways that are not in use.
- Mitigate DDoS attacks. Use Standard protection for critical workloads where outage would have business impact. Also consider CDN as another layer of protection.
- Develop processes and procedures to prevent direct internet access of virtual machines (such as proxy or firewall) with logging and monitoring to enforce policies.
- Implement an automated and gated CI/CD deployment process.

Public endpoints

A public endpoint receives traffic over the internet. The endpoints make the service easily accessible to attackers.

Service Endpoints and Private Link can be leveraged to restrict access to PaaS endpoints only from authorized virtual networks, effectively mitigating data intrusion risks and associated impact to application availability. Service Endpoints provide service level access to a PaaS service, while Private Link provides direct access to a specific PaaS resource to mitigate data exfiltration risks such as malicious admin scenarios.

Configure service endpoints and private links where appropriate.

Are all public endpoints of this workload protected?

An initial design decision is to assess whether you need a public endpoint at all. If you do, protect it by using these mechanisms.

For more information, see Virtual Network service endpoints and What is Azure Private Endpoint?

Web application firewalls (WAFs)

WAFs provide a basic level of security for web applications. WAFs are appropriate if the organizations that have invested in application security as WAFs provide additional defense-in-depth mitigation.

WAFs mitigate the risk of an attacker to exploit commonly seen security vulnerabilities for applications. WAFs provide a basic level of security for web applications. This mechanism is an important mitigation because attackers target web applications for an ingress point into an organization (similar to a client endpoint).

External application endpoints should be protected against common attack vectors, from Denial of Service (DoS) attacks like Slowloris to app-level exploits, to prevent potential application downtime due to malicious intent. Azure-native technologies such as Azure Firewall, Application Gateway/Azure Front Door, WAF, and DDoS Network Protection can be used to achieve requisite protection (Azure DDoS Protection).

Azure Application Gateway has WAF capabilities to inspect web traffic and detect attacks at the HTTP layer. It's a load balancer and HTTP(S) full reverse proxy that can do secure socket layer (SSL) encryption and decryption.

For example, your workload is hosted in Application Service Environments(ILB ASE). The APIs are consolidated internally and exposed to external users. This external exposure could be achieved using an Application Gateway. This service is a load balancer. It forwards request to the internal API Management service, which in turn consumes the APIs deployed in the ASE. Application Gateway is also configured over port 443 for secured and reliable outbound calls.

> **Tip**
>
> The design considerations for the preceding example are described in **Publishing internal APIs to external users**.

Azure Front Door and Azure Content Delivery Network (CDN) also have WAF capabilities.

Suggestion actions

Protect all public endpoints with appropriate solutions such as Azure Front Door, Application Gateway, Azure Firewall, Azure DDOS Protection, or any third-party solution.

Learn more

- What is Azure Firewall?
- Azure DDoS Protection overview
- Azure Front Door documentation
- What is Azure Application Gateway?

Azure Firewall

Protect the entire virtual network against potentially malicious traffic from the internet and other external locations. It inspects incoming traffic and only passes the allowed requests to pass through.

A common design is to implement a DMZ or a perimeter network in front of the application. The DMZ is a separate subnet with the firewall.

> **Tip**
>
> The design considerations are described in **Deploy highly available NVAs**.

Combination approach

When you want higher security and there's a mix of web and non-web workloads in the virtual network use both Azure Firewall and Application Gateway. There are several ways in which those two services can work together.

For example, you want to filter egress traffic. You want to allow connectivity to a specific Azure Storage Account but not others. You'll need fully qualified domain name (FQDN)-based filters. In this case run Firewall and Application Gateway in parallel.

Another popular design is when you want Azure Firewall to inspect all traffic and WAF to protect web traffic, and the application needs to know the client's source IP address. In this case, place Application Gateway in front of Firewall. Conversely, you can place

Firewall in front of WAF if you want to inspect and filter traffic before it reaches the Application Gateway.

For more information, see Firewall and Application Gateway for virtual networks.

It's challenging to write concise firewall rules for networks where different cloud resources dynamically spin up and down. Use Microsoft Defender for Cloud to detect misconfiguration risks.

Authentication

Disable insecure legacy protocols for internet-facing services. Legacy authentication methods are among the top attack vectors for cloud-hosted services. Those methods don't support other factors beyond passwords and are prime targets for password spraying, dictionary, or brute force attacks.

Mitigate DDoS attacks

In a distributed denial-of-service (DDoS) attack, the server is overloaded with fake traffic. DDoS attacks are common and can be debilitating. An attack can completely block access or take down services. Make sure all business-critical web application and services have DDoS mitigation beyond the default defenses so that the application doesn't experience downtime because that can negatively impact business.

Microsoft recommends adopting advanced protection for any services where downtime will have negative impact on the business.

How do you implement DDoS protection?

Here are some considerations:

- DDoS protection at the infrastructure level in which your workload runs. Azure infrastructure has built-in defenses for DDoS attacks.
- DDoS protection at the network (layer 3) layer. Azure provides additional protection for services provisioned in a virtual network.
- DDoS protection with caching. Content delivery network (CDN) can add another layer of protection. In a DDoS attack, a CDN intercepts the traffic and stops it from reaching the backend server. Azure CDN is natively protected. Azure also supports popular CDNs that are protected with proprietary DDoS mitigation platform.
- Advanced DDoS protection. In your security baseline, consider features with monitoring techniques that use machine learning to detect anomalous traffic and proactively protect your application before service degradation occurs.

For information about Azure DDoS Protection services, see Azure DDoS Protection documentation.

Suggested action

Identify critical workloads that are susceptible to DDoS attacks and enable Distributed Denial of Service (DDoS) mitigations for all business-critical web applications and services.

Learn more

For a list of reference architectures that demonstrate the use of DDoS protection, see Azure DDoS Protection reference architectures.

Adopt DevOps

Developers shouldn't publish their code directly to app servers.

Does the organization have an CI/CD process for publishing code in this workload?

Implement lifecycle of continuous integration, continuous delivery (CI/CD) for applications. Have processes and tools in place that aid in an automated and gated CI/CD deployment process.

How are the publishing methods secured?

Application resources allowing multiple methods to publish app content, such as FTP, Web Deploy should have the unused endpoints disabled. For Azure Web Apps, SCM is the recommended endpoint. It can be protected separately with network restrictions for sensitive use cases.

Next step

Data flow

Related links

- Azure Firewall
- What is Azure Web Application Firewall on Azure Application Gateway?
- Azure DDoS Protection

> Go back to the main article: Network security

Traffic flow security in Azure

Article • 11/30/2022

Protect data anywhere it goes including cloud services, mobile devices, workstations, or collaboration platforms. In addition to using access control and encryption mechanisms, apply strong network controls that detect, monitor, and contain attacks.

Key points

- Control network traffic between subnets (east-west) and application tiers (north-south).
- Apply a layered defense-in-depth approach that starts with Zero-Trust policies.
- Use a cloud application security broker (CASB).

East-west and north-south traffic

When analyzing the network flow of a workload, distinguish between east-west traffic from north-south traffic. Most cloud architectures use a combination of both types.

Is the traffic between subnets, Azure components and tiers of the workload managed and secured?

- **North-south traffic**

  North-south refers to the traffic that flows in and out of a datacenter. For example, traffic from an application to a backend service. This type of traffic is a typical target for attack vectors because it flows over the public internet. Proper network controls must be in place so that the queries to and from a data center are secure.

  Consider a typical flow in an Azure Kubernetes Service (AKS) cluster. The cluster receives incoming (ingress) traffic from HTTP requests. The cluster can also send outgoing (egress) traffic to send queries to other services, such as pulling a container image.

  Your design can use Web Application Firewall on Application Gateway to secure ingress traffic, and Azure Firewall to secure outgoing (egress) traffic.

- **East-west traffic**

  East-west traffic refers to traffic between or within data centers. For this type of traffic, several resources of the network infrastructure communicate with each other. Those resources can be virtual networks, subnets within those virtual networks, and so on. Security of east-west traffic can get overlooked even though it makes up a large portion of the workload traffic. It's assumed that the infrastructure firewalls are sufficient to block attacks. Make sure there are proper controls between network resources.

Extending the example of the AKS cluster to this concept, east-west traffic is the traffic within the cluster. For example, communication between pods, such as the ingress controller and the workload. If your workload is composed of multiple applications, the communication between those applications would fall into this category.

By using Kubernetes network policies, you can restrict which pods can communicate, starting from a Zero-Trust policy and then opening specific communication paths as needed.

> **Tip**
>
> Here are the resources for the preceding AKS example:
>
> **GitHub: Azure Kubernetes Service (AKS) Secure Baseline Reference Implementation**.

> The design considerations are described in Azure Kubernetes Service (AKS) production baseline.

Data exfiltration

Data exfiltration is a common attack where an internal or external malicious actor does an unauthorized data transfer. Most often access is gained because of lack of network controls.

Network virtual appliance (NVA) solutions and Azure Firewall (for supported protocols) can be leveraged as a reverse proxy to restrict access to only authorized PaaS services for services where Private Link is not yet supported (Azure Firewall).

Configure Azure Firewall or a third-party next generation firewall to protect against data exfiltration concerns.

Are there controls in the workload design to detect and protect from data exfiltration?

Choose a defense-in-depth design that can protect network communications at various layers, such as a hub-spoke topology. Azure provides several controls to support the layered design:

- Use Azure Firewall to allow or deny traffic using layer 3 to layer 7 controls.
- Use Azure Virtual Network User Defined Routes (UDR) to control next hop for traffic.
- Control traffic with Network Security Groups (NSGs) between resources within a virtual network, internet, and other virtual networks.
- Secure the endpoints through Azure PrivateLink and Private Endpoints.
- Detect and protect at deep levels through packet inspection.
- Detect attacks and respond to alerts through Microsoft Sentinel and Microsoft Defender for Cloud.

> ⓘ **Important**
>
> Network controls are not sufficient in blocking data exfiltration attempts. Harden the protection with proper identity controls, key protection, and encryption. For more information, see these sections:
>
> - **Data protection considerations**
> - **Identity and access management considerations**

Have you considered a cloud application security broker (CASB) for this workload?

CASBs provide a central point of control for enforcing policies. They provide rich visibility, control over data travel, and sophisticated analytics to identify and combat cyberthreats across all Microsoft and third-party cloud services.

Learn more

- Azure firewall documentation
- Azure Marketplace networking apps

Related links

- Azure Firewall
- Network Security Groups (NSG)
- What is Azure Web Application Firewall on Azure Application Gateway?
- What is Azure Private Link?

> Go back to the main article: Network security

Azure best practices for network security

Article • 03/17/2023

This article discusses a collection of Azure best practices to enhance your network security. These best practices are derived from our experience with Azure networking and the experiences of customers like yourself.

For each best practice, this article explains:

- What the best practice is
- Why you want to enable that best practice
- What might be the result if you fail to enable the best practice
- Possible alternatives to the best practice
- How you can learn to enable the best practice

These best practices are based on a consensus opinion, and Azure platform capabilities and feature sets, as they exist at the time this article was written. Opinions and technologies change over time and this article will be updated regularly to reflect those changes.

Use strong network controls

You can connect Azure virtual machines (VMs) and appliances to other networked devices by placing them on Azure virtual networks. That is, you can connect virtual network interface cards to a virtual network to allow TCP/IP-based communications between network-enabled devices. Virtual machines connected to an Azure virtual network can connect to devices on the same virtual network, different virtual networks, the internet, or your own on-premises networks.

As you plan your network and the security of your network, we recommend that you centralize:

- Management of core network functions like ExpressRoute, virtual network and subnet provisioning, and IP addressing.
- Governance of network security elements, such as network virtual appliance functions like ExpressRoute, virtual network and subnet provisioning, and IP addressing.

If you use a common set of management tools to monitor your network and the security of your network, you get clear visibility into both. A straightforward, unified security strategy reduces errors because it increases human understanding and the reliability of automation.

Logically segment subnets

Azure virtual networks are similar to LANs on your on-premises network. The idea behind an Azure virtual network is that you create a network, based on a single private IP address space, on which you can place all your Azure virtual machines. The private IP address spaces available are in the Class A (10.0.0.0/8), Class B (172.16.0.0/12), and Class C (192.168.0.0/16) ranges.

Best practices for logically segmenting subnets include:

Best practice: Don't assign allow rules with broad ranges (for example, allow 0.0.0.0 through 255.255.255.255).
Detail: Ensure troubleshooting procedures discourage or ban setting up these types of rules. These allow rules lead to a false sense of security and are frequently found and exploited by red teams.

Best practice: Segment the larger address space into subnets.
Detail: Use CIDR -based subnetting principles to create your subnets.

Best practice: Create network access controls between subnets. Routing between subnets happens automatically, and you don't need to manually configure routing tables. By default, there are no network access controls between the subnets that you create on an Azure virtual network.
Detail: Use a network security group to protect against unsolicited traffic into Azure subnets. Network security groups (NSGs) are simple, stateful packet inspection devices. NSGs use the 5-tuple approach (source IP, source port, destination IP, destination port, and layer 4 protocol) to create allow/deny rules for network traffic. You allow or deny traffic to and from a single IP address, to and from multiple IP addresses, or to and from entire subnets.

When you use network security groups for network access control between subnets, you can put resources that belong to the same security zone or role in their own subnets.

Best practice: Avoid small virtual networks and subnets to ensure simplicity and flexibility. **Detail**: Most organizations add more resources than initially planned, and reallocating addresses is labor intensive. Using small subnets adds limited security value, and mapping a network security group to each subnet adds overhead. Define subnets broadly to ensure that you have flexibility for growth.

Best practice: Simplify network security group rule management by defining Application Security Groups.
Detail: Define an Application Security Group for lists of IP addresses that you think might change in the future or be used across many network security groups. Be sure to name Application Security Groups clearly so others can understand their content and purpose.

Adopt a Zero Trust approach

Perimeter-based networks operate on the assumption that all systems within a network can be trusted. But today's employees access their organization's resources from anywhere on various devices and apps, which makes perimeter security controls irrelevant. Access control policies that focus only on who can access a resource aren't enough. To master the balance between security and productivity, security admins also need to factor in *how* a resource is being accessed.

Networks need to evolve from traditional defenses because networks might be vulnerable to breaches: an attacker can compromise a single endpoint within the trusted boundary and then quickly expand a foothold across the entire network. Zero Trust networks eliminate the concept of trust based on network location within a perimeter. Instead, Zero Trust architectures use device and user trust claims to gate access to organizational data and resources. For new initiatives, adopt Zero Trust approaches that validate trust at the time of access.

Best practices are:

Best practice: Give Conditional Access to resources based on device, identity, assurance, network location, and more.
Detail: Azure AD Conditional Access lets you apply the right access controls by implementing automated access control decisions based on the required conditions. For more information, see Manage access to Azure management with Conditional Access.

Best practice: Enable port access only after workflow approval.
Detail: You can use just-in-time VM access in Microsoft Defender for Cloud to lock down inbound traffic to your Azure VMs, reducing exposure to attacks while providing easy access to connect to VMs when needed.

Best practice: Grant temporary permissions to perform privileged tasks, which prevents malicious or unauthorized users from gaining access after the permissions have expired. Access is granted only when users need it.
Detail: Use just-in-time access in Azure AD Privileged Identity Management or in a third-party solution to grant permissions to perform privileged tasks.

Zero Trust is the next evolution in network security. The state of cyberattacks drives organizations to take the "assume breach" mindset, but this approach shouldn't be limiting. Zero Trust networks protect corporate data and resources while ensuring that organizations can build a modern workplace by using technologies that empower employees to be productive anytime, anywhere, in any way.

Control routing behavior

When you put a virtual machine on an Azure virtual network, the VM can connect to any other VM on the same virtual network, even if the other VMs are on different subnets. This is possible because a collection of system routes enabled by default allows this type of communication. These default routes allow VMs on the same virtual network to initiate connections with each other, and with the internet (for outbound communications to the internet only).

Although the default system routes are useful for many deployment scenarios, there are times when you want to customize the routing configuration for your deployments. You can configure the next-hop address to reach specific destinations.

We recommend that you configure user-defined routes when you deploy a security appliance for a virtual network. We talk about this recommendation in a later section titled secure your critical Azure service resources to only your virtual networks.

> ⓘ **Note**
>
> User-defined routes aren't required, and the default system routes usually work.

Use virtual network appliances

Network security groups and user-defined routing can provide a certain measure of network security at the network and transport layers of the OSI model. But in some situations, you want or need to enable security at high levels of the stack. In such situations, we recommend that you deploy virtual network security appliances provided by Azure partners.

Azure network security appliances can deliver better security than what network-level controls provide. Network security capabilities of virtual network security appliances include:

- Firewalling
- Intrusion detection/intrusion prevention

- Vulnerability management
- Application control
- Network-based anomaly detection
- Web filtering
- Antivirus
- Botnet protection

To find available Azure virtual network security appliances, go to the Azure Marketplace and search for "security" and "network security."

Deploy perimeter networks for security zones

A perimeter network (also known as a DMZ) is a physical or logical network segment that provides an extra layer of security between your assets and the internet. Specialized network access control devices on the edge of a perimeter network allow only desired traffic into your virtual network.

Perimeter networks are useful because you can focus your network access control management, monitoring, logging, and reporting on the devices at the edge of your Azure virtual network. A perimeter network is where you typically enable distributed denial of service (DDoS) protection, intrusion detection/intrusion prevention systems (IDS/IPS), firewall rules and policies, web filtering, network antimalware, and more. The network security devices sit between the internet and your Azure virtual network and have an interface on both networks.

Although this is the basic design of a perimeter network, there are many different designs, like back-to-back, tri-homed, and multi-homed.

Based on the Zero Trust concept mentioned earlier, we recommend that you consider using a perimeter network for all high security deployments to enhance the level of network security and access control for your Azure resources. You can use Azure or a third-party solution to provide an extra layer of security between your assets and the internet:

- Azure native controls. Azure Firewall and Azure Web Application Firewall offer basic security advantages. Advantages are a fully stateful firewall as a service, built-in high availability, unrestricted cloud scalability, FQDN filtering, support for OWASP core rule sets, and simple setup and configuration.
- Third-party offerings. Search the Azure Marketplace for next-generation firewall (NGFW) and other third-party offerings that provide familiar security tools and enhanced levels of network security. Configuration might be more complex, but a third-party offering might allow you to use existing capabilities and skillsets.

Avoid exposure to the internet with dedicated WAN links

Many organizations have chosen the hybrid IT route. With hybrid IT, some of the company's information assets are in Azure, and others remain on-premises. In many cases, some components of a service are running in Azure while other components remain on-premises.

In a hybrid IT scenario, there's usually some type of cross-premises connectivity. Cross-premises connectivity allows the company to connect its on-premises networks to Azure virtual networks. Two cross-premises connectivity solutions are available:

- Site-to-site VPN. It's a trusted, reliable, and established technology, but the connection takes place over the internet. Bandwidth is constrained to a maximum of about 1.25 Gbps. Site-to-site VPN is a desirable option in some scenarios.
- **Azure ExpressRoute**. We recommend that you use ExpressRoute for your cross-premises connectivity. ExpressRoute lets you extend your on-premises networks into the Microsoft cloud over a private connection facilitated by a connectivity provider. With ExpressRoute, you can establish connections to Microsoft cloud services like Azure, Microsoft 365, and Dynamics 365. ExpressRoute is a dedicated WAN link between your on-premises location or a Microsoft Exchange hosting provider. Because this is a telco connection, your data doesn't travel over the internet, so it isn't exposed to the potential risks of internet communications.

The location of your ExpressRoute connection can affect firewall capacity, scalability, reliability, and network traffic visibility. You'll need to identify where to terminate ExpressRoute in existing (on-premises) networks. You can:

- Terminate outside the firewall (the perimeter network paradigm). Use this recommendation if you require visibility into the traffic, if you need to continue an existing practice of isolating datacenters, or if you're solely putting extranet resources on Azure.
- Terminate inside the firewall (the network extension paradigm). This is the default recommendation. In all other cases, we recommend treating Azure as another datacenter.

Optimize uptime and performance

If a service is down, information can't be accessed. If performance is so poor that the data is unusable, you can consider the data to be inaccessible. From a security perspective, you need to do whatever you can to make sure that your services have optimal uptime and performance.

A popular and effective method for enhancing availability and performance is load balancing. Load balancing is a method of distributing network traffic across servers that are part of a service. For example, if you have front-end web servers as part of your service, you can use load balancing to distribute the traffic across your multiple front-end web servers.

This distribution of traffic increases availability because if one of the web servers becomes unavailable, the load balancer stops sending traffic to that server and redirects it to the servers that are still online. Load balancing also helps performance, because the processor, network, and memory overhead for serving requests is distributed across all the load-balanced servers.

We recommend that you employ load balancing whenever you can, and as appropriate for your services. Following are scenarios at both the Azure virtual network level and the global level, along with load-balancing options for each.

Scenario: You have an application that:

- Requires requests from the same user/client session to reach the same back-end virtual machine. Examples of this are shopping cart apps and web mail servers.
- Accepts only a secure connection, so unencrypted communication to the server isn't an acceptable option.
- Requires multiple HTTP requests on the same long-running TCP connection to be routed or load balanced to different back-end servers.

Load-balancing option: Use Azure Application Gateway, an HTTP web traffic load balancer. Application Gateway supports end-to-end TLS encryption and TLS termination at the gateway. Web servers can then be unburdened from encryption and decryption overhead and traffic flowing unencrypted to the back-end servers.

Scenario: You need to load balance incoming connections from the internet among your servers located in an Azure virtual network. Scenarios are when you:

- Have stateless applications that accept incoming requests from the internet.
- Don't require sticky sessions or TLS offload. Sticky sessions is a method used with Application Load Balancing, to achieve server-affinity.

Load-balancing option: Use the Azure portal to create an external load balancer that spreads incoming requests across multiple VMs to provide a higher level of availability.

Scenario: You need to load balance connections from VMs that are not on the internet. In most cases, the connections that are accepted for load balancing are initiated by devices on an Azure virtual network, such as SQL Server instances or internal web servers.
Load-balancing option: Use the Azure portal to create an internal load balancer that spreads incoming requests across multiple VMs to provide a higher level of availability.

Scenario: You need global load balancing because you:

- Have a cloud solution that is widely distributed across multiple regions and requires the highest level of uptime (availability) possible.
- Need the highest level of uptime possible to make sure that your service is available even if an entire datacenter becomes unavailable.

Load-balancing option: Use Azure Traffic Manager. Traffic Manager makes it possible to load balance connections to your services based on the location of the user.

For example, if the user makes a request to your service from the EU, the connection is directed to your services located in an EU datacenter. This part of Traffic Manager global load balancing helps to improve performance because connecting to the nearest datacenter is faster than connecting to datacenters that are far away.

Disable RDP/SSH Access to virtual machines

It's possible to reach Azure virtual machines by using Remote Desktop Protocol (RDP) and the Secure Shell (SSH) protocol. These protocols enable the management VMs from remote locations and are standard in datacenter computing.

The potential security problem with using these protocols over the internet is that attackers can use brute force techniques to gain access to Azure virtual machines. After the attackers gain access, they can use your VM as a launch point for compromising other machines on your virtual network or even attack networked devices outside Azure.

We recommend that you disable direct RDP and SSH access to your Azure virtual machines from the internet. After direct RDP and SSH access from the internet is disabled, you have other options that you can use to access these VMs for remote management.

Scenario: Enable a single user to connect to an Azure virtual network over the internet.
Option: Point-to-site VPN is another term for a remote access VPN client/server connection. After the point-to-site connection is established, the user can use RDP or SSH to connect to any VMs located on the Azure virtual network that the user

connected to via point-to-site VPN. This assumes that the user is authorized to reach those VMs.

Point-to-site VPN is more secure than direct RDP or SSH connections because the user has to authenticate twice before connecting to a VM. First, the user needs to authenticate (and be authorized) to establish the point-to-site VPN connection. Second, the user needs to authenticate (and be authorized) to establish the RDP or SSH session.

Scenario: Enable users on your on-premises network to connect to VMs on your Azure virtual network.
Option: A site-to-site VPN connects an entire network to another network over the internet. You can use a site-to-site VPN to connect your on-premises network to an Azure virtual network. Users on your on-premises network connect by using the RDP or SSH protocol over the site-to-site VPN connection. You don't have to allow direct RDP or SSH access over the internet.

Scenario: Use a dedicated WAN link to provide functionality similar to the site-to-site VPN.
Option: Use ExpressRoute. It provides functionality similar to the site-to-site VPN. The main differences are:

- The dedicated WAN link doesn't traverse the internet.
- Dedicated WAN links are typically more stable and perform better.

Secure your critical Azure service resources to only your virtual networks

Use Azure Private Link to access Azure PaaS Services (for example, Azure Storage and SQL Database) over a private endpoint in your virtual network. Private Endpoints allow you to secure your critical Azure service resources to only your virtual networks. Traffic from your virtual network to the Azure service always remains on the Microsoft Azure backbone network. Exposing your virtual network to the public internet is no longer necessary to consume Azure PaaS Services.

Azure Private Link provides the following benefits:

- **Improved security for your Azure service resources**: With Azure Private Link, Azure service resources can be secured to your virtual network using private endpoint. Securing service resources to a private endpoint in virtual network provides improved security by fully removing public internet access to resources, and allowing traffic only from private endpoint in your virtual network.

- **Privately access Azure service resources on the Azure platform**: Connect your virtual network to services in Azure using private endpoints. There's no need for a public IP address. The Private Link platform will handle the connectivity between the consumer and services over the Azure backbone network.
- **Access from On-premises and peered networks**: Access services running in Azure from on-premises over ExpressRoute private peering, VPN tunnels, and peered virtual networks using private endpoints. There's no need to configure ExpressRoute Microsoft peering or traverse the internet to reach the service. Private Link provides a secure way to migrate workloads to Azure.
- **Protection against data leakage**: A private endpoint is mapped to an instance of a PaaS resource instead of the entire service. Consumers can only connect to the specific resource. Access to any other resource in the service is blocked. This mechanism provides protection against data leakage risks.
- **Global reach**: Connect privately to services running in other regions. The consumer's virtual network could be in region A and it can connect to services in region B.
- **Simple to set up and manage**: You no longer need reserved, public IP addresses in your virtual networks to secure Azure resources through an IP firewall. There are no NAT or gateway devices required to set up the private endpoints. Private endpoints are configured through a simple workflow. On service side, you can also manage the connection requests on your Azure service resource with ease. Azure Private Link works for consumers and services belonging to different Azure Active Directory tenants too.

To learn more about private endpoints and the Azure services and regions that private endpoints are available for, see Azure Private Link.

Next steps

See Azure security best practices and patterns for more security best practices to use when you're designing, deploying, and managing your cloud solutions by using Azure.

Data protection considerations

Article • 12/27/2022

Classify, protect, and monitor sensitive data assets using access control, encryption, and logging in Azure. Provide controls on data at rest and in transit.

Checklist

How are you managing encryption for this workload?

- ✓ Use identity based storage access controls.
- ✓ Use built-in features for data encryption for Azure services.
- ✓ Classify all stored data and encrypt it.
- ✓ Protect data moving over a network through encryption at all points so that it's not accessed unauthorized users.
- ✓ Store keys in managed key vault service with identity-based access control and audit policies.
- ✓ Rotate keys and other secrets frequently.

Azure security benchmark

The Azure Security Benchmark includes a collection of high-impact security recommendations you can use to help secure the services you use in Azure:

> The questions in this section are aligned to the Azure Security Benchmarks Data Protection.

Reference architecture

Here are some reference architectures related to secure storage:

- Using Azure file shares in a hybrid environment
- DevSecOps in Azure

Next steps

We recommend that you review the practices and tools implemented as part of the development cycle.

Design Storage Encryption

Related links

> Back to the main article: Security

Data encryption in Azure

Article • 11/30/2022

Data can be categorized by its state:

- **Data at rest**. All information storage objects, containers, and types that exist statically on physical media, whether magnetic or optical disk.

- **Data in transit**. Data that is being transferred between components, locations, or programs.

In a cloud solution, a single business transaction can lead to multiple data operations where data moves from one storage medium to another. To provide complete data protection, it must be encrypted on storage volumes and while it's transferred from one point to another.

Key points

- Use identity-based storage access controls.
- Use standard and recommended encryption algorithms.
- Use only secure hash algorithms (SHA-2 family).
- Classify your data at rest and use encryption.
- Encrypt virtual disks.
- Use an additional key encryption key (KEK) to protect your data encryption key (DEK).
- Protect data in transit through encrypted network channels (TLS/HTTPS) for all client/server communication. Use TLS 1.2 on Azure.

Azure encryption features

Azure provides built-in features for data encryption in many layers that participate in data processing. We recommend that for each service, enable the encryption capability. The encryption is handled automatically using Azure-managed keys. This almost requires no user interaction.

We recommend implementing identity-based storage access controls. Authentication with a shared key (like a Shared Access Signature) doesn't permit the same flexibility and control as identity-based access control. The leak of a shared key might allow indefinite access to a resource, whereas a role-based access control can be identified and authenticated more strongly.

Storage in a cloud service like Azure is architected and implemented quite differently than on-premises solutions to enable massive scaling, modern access through REST APIs, and isolation between tenants. Cloud service providers make multiple methods of access control over storage resources available. Examples include shared keys, shared signatures, anonymous access, and identity provider-based methods.

Consider some built-in features of Azure Storage:

- **Identity-based access**. Supports access through Azure Active Directory (Azure AD) and key-based authentication mechanisms, such as Symmetric Shared Key Authentication, or Shared Access Signature (SAS).
- **Built-in encryption**. All stored data is encrypted by Azure storage. Data cannot be read by a tenant if it has not been written by that tenant. This feature provides control over cross tenant data leakage.
- **Region-based controls**. Data remains only in the selected region and three synchronous copies of data are maintained within that region. Azure storage provides detailed activity logging is available on an opt-in basis.
- **Firewall features**. The firewall provides an additional layer of access control and storage threat protection to detect anomalous access and activities.

For the complete set of features, see Azure Storage Service encryption.

Suggested action

Identify provider methods of authentication and authorization that are the least likely to be compromised, and enable more fine-grained role-based access controls over storage resources.

Learn more

For more information, reference Authorize access to blobs using Azure Active Directory.

Standard encryption algorithms

Does the organization use industry standard encryption algorithms instead of creating their own?

Organizations should not develop and maintain their own encryption algorithms. Avoid using custom encryption algorithms or direct cryptography in your workload. These methods rarely stand up to real world attacks.

Secure standards already exist on the market and should be preferred. If custom implementation is required, developers should use well-established cryptographic algorithms and secure standards. Use Advanced Encryption Standard (AES) as a symmetric block cipher, AES-128, AES-192, and AES-256 are acceptable.

Developers should use cryptography APIs built into operating systems instead of non-platform cryptography libraries. For .NET, follow the .NET Cryptography Model.

We advise using standard and recommended encryption algorithms.

For more information, refer to Choose an algorithm.

Are modern hashing functions used?

Applications should use the SHA-2 family of hash algorithms (SHA-256, SHA-384, SHA-512).

Data at rest

All important data should be classified and encrypted with an encryption standard. Classify and protect all information storage objects. Use encryption to make sure the contents of files cannot be accessed by unauthorized users.

Data at rest is encrypted by default in Azure, but is your critical data classified and tagged, or labeled so that it can be audited?

Your most sensitive data might include business, financial, healthcare, or personal information. Discovering and classifying this data can play a pivotal role in your organization's information protection approach. It can serve as infrastructure for:

- Helping to meet standards for data privacy and requirements for regulatory compliance.
- Various security scenarios, such as monitoring (auditing) and alerting on anomalous access to sensitive data.
- Controlling access to and hardening the security of databases that contain highly sensitive data.

Suggested action

Classify your data. Consider using Data Discovery & Classification in Azure SQL Database.

Data classification

A crucial initial exercise for protecting data is to organize it into categories based on certain criteria. The classification criteria can be your business needs, compliance requirements, and the type of data.

Depending on the category, you can protect it through:

- Standard encryption mechanisms.
- Enforce security governance through policies.
- Conduct audits to make sure the security measures are compliant.

One way of classifying data is through the use of tags.

Does the organization encrypt virtual disk files for virtual machines that are associated with this workload?

There are many options to store files in the cloud. Cloud-native apps typically use Azure Storage. Apps that run on VMs use them to store files. VMs use virtual disk files as virtual storage volumes and exist in a blob storage.

Consider a hybrid solution. Files can move from on-premises to the cloud, from the cloud to on-premises, or between services hosted in the cloud. One strategy is to make sure that the files and their contents aren't accessible to unauthorized users. You can use authentication-based access controls to prevent unauthorized downloading of files. However, that is not enough. Have a backup mechanism to secure the virtual disk files in case authentication and authorization or its configuration is compromised. There are several approaches. You can encrypt the virtual disk files. If an attempt is made to mount disk files, the contents of the files cannot be accessed because of the encryption.

We recommend that you enable virtual disk encryption. For information about how to encrypt Windows VM disks, see Quickstart: Create and encrypt a Windows VM with the Azure CLI.

Azure-based virtual disks are stored as files in a Storage account. If no encryption is applied to a virtual disk, and an attacker manages to download a virtual disk image file, it can be mounted and inspected at the attacker's leisure as if they had physical access to the source computer. Encrypting virtual disk files helps prevent attackers from gaining access to the contents of those disk files in the event they are able to download them. Depending on the sensitivity of the information stored on the disk, unencrypted access could represent a critical risk to confidential business data (such as a SQL database) or identity (such as an AD Domain Controller).

An example of virtual disk encryption is Azure Disk Encryption.

Azure Disk Encryption helps protect and safeguard your data to meet your organizational security and compliance commitments. It uses the Bitlocker-feature of Windows (or DM-Crypt on Linux) to provide volume encryption for the OS and data disks of Azure virtual machines (VMs). It is integrated with Azure Key Vault to help you control and manage the disk encryption keys, and secrets.

Virtual machines use virtual disk files as storage volumes and exist in a cloud service provider's blob storage system. These files can be moved from on-premises to cloud systems, from cloud systems to on-premises, or between cloud systems. Due to the mobility of these files, it's recommended that the files and the contents are not accessible to unauthorized users.

Does the organization use identity-based storage access controls for this workload?

There are many ways to control access to data: shared keys, shared signatures, anonymous access, identity provider-based. Use Azure Active Directory (Azure AD) and role-based access control (RBAC) to grant access. For more information, see Identity and access management considerations.

Does the organization protect keys in this workload with an additional key encryption key (KEK)?

Use more than one encryption key in an encryption at rest implementation. Storing an encryption key in Azure Key Vault ensures secure key access and central management of keys.

Use an additional key encryption key (KEK) to protect your data encryption key (DEK).

Suggested actions

Identify unencrypted virtual machines via Microsoft Defender for Cloud or script, and encrypt via Azure Disk Encryption. Ensure all new virtual machines are encrypted by default and regularly monitor for unprotected disks.

Learn more

Azure Disk Encryption for virtual machines and virtual machine scale sets

Data in transit

Data in transit should be encrypted at all points to ensure data integrity.

Protecting data in transit should be an essential part of your data protection strategy. Because data is moving back and forth from many locations, we generally recommend that you always use SSL/TLS protocols to exchange data across different locations.

For data moving between your on-premises infrastructure and Azure, consider appropriate safeguards such as HTTPS or VPN. When sending encrypted traffic between an Azure virtual network and an on-premises location over the public internet, use Azure VPN Gateway.

Does the workload communicate over encrypted network traffic only?

Any network communication between client and server where man-in-the-middle attacks can occur, must be encrypted. All website communication should use HTTPS, no matter the perceived sensitivity of transferred data. Man-in-the-middle attacks can occur anywhere on the site, not just login forms.

This mechanism can be applied to use cases such as:

- Web applications and APIs for all communication with clients.
- Data moving across a service bus from on-premises to the cloud and other way around, or during an input/output process.

In certain architecture styles such as microservices, data must be encrypted during communication between the services.

What TLS version is used across workloads?

Using the latest version of TLS is preferred. All Azure services support TLS 1.2 on public HTTPS endpoints. Migrate solutions to support TLS 1.2 and use this version by default.

When traffic from clients using older versions of TLS is minimal, or it's acceptable to fail requests made with an older version of TLS, consider enforcing a minimum TLS version. For information about TLS support in Azure Storage, see Remediate security risks with a minimum version of TLS.

Sometimes you need to isolate your entire communication channel between your on-premises and the cloud infrastructure by using either a virtual private network (VPN) or ExpressRoute. For more information, see these articles:

- Extending on-premises data solutions to the cloud
- Configure a Point-to-Site VPN connection to a VNet using native Azure certificate authentication: Azure portal

For more information, see Protect data in transit.

Is there any portion of the application that doesn't secure data in transit?

All data should be encrypted in transit using a common encryption standard. Determine if all components in the solution are using a consistent standard. There are times when encryption is not possible because of technical limitations, make sure the reason is clear and valid.

Suggested actions

Identify workloads using unencrypted sessions and configure the service to require encryption.

Learn more

- Encrypt data in transit
- Azure encryption overview

Next steps

While it's important to protect data through encryption, it's equally important to protect they keys that provide access to the data.

Key and secret management

Related links

Identity and access management services authenticate and grant permission to users, partners, customers, applications, services, and other entities. For security considerations, see Azure identity and access management considerations.

> Back to the main article: Data protection

Key and secret management considerations in Azure

Article • 11/30/2022

Encryption is an essential tool for security because it restricts access. However, it's equally important to protect the secrets (keys, certificates) key that provide access to the data.

Key points

- Use identity-based access control instead of cryptographic keys.
- Use standard and recommended encryption algorithms.
- Store keys and secrets in managed key vault service. Control permissions with an access model.
- Rotate keys and other secrets frequently. Replace expired or compromised secrets.

Identity-based access control

Organizations shouldn't develop and maintain their own encryption algorithms. There are many ways to provide access control over storage resources available, such as:

- Shared keys
- Shared signatures
- Anonymous access
- Identity provider-based methods

Secure standards already exist on the market and should be preferred. AES should be used as symmetric block cipher, `AES-128`, `AES-192`, and `AES-256` are acceptable. Crypto APIs built into operating systems should be used where possible, instead of non-platform crypto libraries. For .NET, make sure you follow the .NET Cryptography Model.

Do you prioritize authentication through identity services for a workload over cryptographic keys?

Protection of cryptographic keys can often get overlooked or implemented poorly. Managing keys securely with application code is especially difficult and can lead to mistakes such as accidentally publishing sensitive access keys to public code repositories.

Use of identity-based options for storage access control is recommended. This option uses role-based access controls (RBAC) over storage resources. Use RBAC to assign permissions to users, groups, and applications at a certain scope. Identity systems such as Azure Active Directory (Azure AD) offer secure and usable experience for access control with built-in mechanisms for handling key rotation, monitoring for anomalies, and others.

> **Note**
>
> Grant access based on the principle of least privilege. Risk of giving more privileges than necessary can lead to data compromise.

Suppose you need to store sensitive data in Azure Blob Storage. You can use Azure AD and RBAC to authenticate a service principal that has the required permissions to access the storage. For more information about the feature, reference Authorize access to blobs and queues using Azure Active Directory.

> **Tip**
>
> Using SAS tokens is a common way to control access. SAS tokens are created by using the service owner's Azure AD credentials. The tokens are created per resource and you can use Azure RBAC to restrict access. SAS tokens have a time limit, which controls the window of exposure. Here are the resources for the preceding example:
>
> **GitHub: Azure Cognitive Services Reference Implementation**.
>
> The design considerations are described in **Speech transcription with Azure Cognitive Services**.

Key storage

To prevent security leaks, store the following keys and secrets in a secure store:

- API keys
- Database connection strings
- Data encryption keys
- Passwords

Sensitive information shouldn't be stored within the application code or configuration. An attacker gaining read access to source code shouldn't gain knowledge of application and environment-specific secrets.

Store all application keys and secrets in a managed key vault service such as Azure Key Vault or HashiCorp Vault. Storing encryption keys in a managed store further limits access. The workload can access the secrets by authenticating against Key Vault by using managed identities. That access can be restricted with Azure RBAC.

Make sure no keys and secrets for any environment types (Dev, Test, or Production) are stored in application configuration files or CI/CD pipelines. Developers can use Visual Studio Connected Services or local-only files to access credentials.

Have processes that periodically detect exposed keys in your application code. An option is Credential Scanner. For information about the configuring task, reference Credential Scanner task.

Do you have an access model for key vaults to grant access to keys and secrets?

To secure access to your key vaults, control permissions to keys and secrets through an access model. For more information, reference Access model overview.

Suggested actions

Consider using Azure Key Vault for secrets and keys.

Operational considerations

Who is responsible for managing keys and secrets in the application context?

Key and certificate rotation is often the cause of application outages. Even Azure has experienced expired certificates. It's critical that the rotation of keys and certificates be scheduled and fully operationalized. The rotation process should be automated and tested to ensure effectiveness. Azure Key Vault supports key rotation and auditing.

Central SecOps team provides guidance on how keys and secrets are managed (governance). Application DevOps team is responsible for managing the application-related keys and secrets.

What types of keys and secrets are used and how are those generated?

The following approaches include:

- Microsoft-managed Keys
- Customer-managed Keys

- Bring Your Own Key

The decision is often driven by security, compliance, and specific data classification requirements. Develop a clear understanding of these requirements to determine the most suitable type of keys.

Are keys and secrets rotated frequently?

To reduce the attack vectors, secrets require rotation and are prone to expiration. The process should be automated and executed without any human interactions. Storing them in a managed store simplifies those operational tasks by handling key rotation.

Replace secrets after they've reached the end of their active lifetime or if they've been compromised. Renewed certificates should also use a new key. Have a process for situations where keys get compromised (leaked) and need to be regenerated on-demand. For example, secrets rotation in SQL Database.

For more information, reference Key Vault Key Rotation.

By using managed identities, you remove the operational overhead for storing the secrets or certificates of service principals.

Are the expiration dates of SSL/TLS certificates monitored and are processes in place to renew them?

A common cause of application outage is expired SSL/TLS certificates.

Avoid outages by tracking the expiration dates of SSL/TLS certificates and renewing them in due time. Ideally, the process should be automated, although this often depends on used certificate authority (CA). If not automated, use alerts to make sure expiration dates don't go unnoticed.

Suggested actions

Implement a process for SSL certificate management and the automated renewal process with Azure Key Vault.

Learn more

Tutorial: Configure certificate auto-rotation in Key Vault

Related content

Identity and access management services authenticate and grant permission to the following groups:

- Users
- Partners
- Customers
- Applications
- Services
- Other entities

For security considerations, reference Azure identity and access management considerations.

> Back to the main article: Data protection

Next steps

Protect data at rest and in transit through encryption. Make sure you use standard encryption algorithms.

Data encryption

Azure encryption overview

Article • 02/23/2023 • 12 minutes to read

This article provides an overview of how encryption is used in Microsoft Azure. It covers the major areas of encryption, including encryption at rest, encryption in flight, and key management with Azure Key Vault. Each section includes links to more detailed information.

Encryption of data at rest

Data at rest includes information that resides in persistent storage on physical media, in any digital format. The media can include files on magnetic or optical media, archived data, and data backups. Microsoft Azure offers a variety of data storage solutions to meet different needs, including file, disk, blob, and table storage. Microsoft also provides encryption to protect Azure SQL Database, Azure Cosmos DB, and Azure Data Lake.

Data encryption at rest is available for services across the software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS) cloud models. This article summarizes and provides resources to help you use the Azure encryption options.

For a more detailed discussion of how data at rest is encrypted in Azure, see Azure Data Encryption-at-Rest.

Azure encryption models

Azure supports various encryption models, including server-side encryption that uses service-managed keys, customer-managed keys in Key Vault, or customer-managed keys on customer-controlled hardware. With client-side encryption, you can manage and store keys on-premises or in another secure location.

Client-side encryption

Client-side encryption is performed outside of Azure. It includes:

- Data encrypted by an application that's running in the customer's datacenter or by a service application.
- Data that is already encrypted when it is received by Azure.

With client-side encryption, cloud service providers don't have access to the encryption keys and cannot decrypt this data. You maintain complete control of the keys.

Server-side encryption

The three server-side encryption models offer different key management characteristics, which you can choose according to your requirements:

- **Service-managed keys**: Provides a combination of control and convenience with low overhead.
- **Customer-managed keys**: Gives you control over the keys, including Bring Your Own Keys (BYOK) support, or allows you to generate new ones.
- **Service-managed keys in customer-controlled hardware**: Enables you to manage keys in your proprietary repository, outside of Microsoft control. This characteristic is called Host Your Own Key (HYOK). However, configuration is complex, and most Azure services don't support this model.

Azure disk encryption

You can protect your managed disks by using Azure Disk Encryption for Linux VMs, which uses DM-Crypt, or Azure Disk Encryption for Windows VMs, which uses Windows BitLocker, to protect both operating system disks and data disks with full volume encryption.

Encryption keys and secrets are safeguarded in your Azure Key Vault subscription. By using the Azure Backup service, you can back up and restore encrypted virtual machines (VMs) that use Key Encryption Key (KEK) configuration.

Azure Storage Service Encryption

Data at rest in Azure Blob storage and Azure file shares can be encrypted in both server-side and client-side scenarios.

Azure Storage Service Encryption (SSE) can automatically encrypt data before it is stored, and it automatically decrypts the data when you retrieve it. The process is completely transparent to users. Storage Service Encryption uses 256-bit Advanced Encryption Standard (AES) encryption, which is one of the strongest block ciphers available. AES handles encryption, decryption, and key management transparently.

Client-side encryption of Azure blobs

You can perform client-side encryption of Azure blobs in various ways.

You can use the Azure Storage Client Library for .NET NuGet package to encrypt data within your client applications prior to uploading it to your Azure storage.

To learn more about and download the Azure Storage Client Library for .NET NuGet package, see Windows Azure Storage 8.3.0.

When you use client-side encryption with Key Vault, your data is encrypted using a one-time symmetric Content Encryption Key (CEK) that is generated by the Azure Storage client SDK. The CEK is encrypted using a Key Encryption Key (KEK), which can be either a symmetric key or an asymmetric key pair. You can manage it locally or store it in Key Vault. The encrypted data is then uploaded to Azure Storage.

To learn more about client-side encryption with Key Vault and get started with how-to instructions, see Tutorial: Encrypt and decrypt blobs in Azure Storage by using Key Vault.

Finally, you can also use the Azure Storage Client Library for Java to perform client-side encryption before you upload data to Azure Storage, and to decrypt the data when you download it to the client. This library also supports integration with Key Vault for storage account key management.

Encryption of data at rest with Azure SQL Database

Azure SQL Database is a general-purpose relational database service in Azure that supports structures such as relational data, JSON, spatial, and XML. SQL Database supports both server-side encryption via the Transparent Data Encryption (TDE) feature and client-side encryption via the Always Encrypted feature.

Transparent Data Encryption

TDE is used to encrypt SQL Server, Azure SQL Database, and Azure Synapse Analytics data files in real time, using a Database Encryption Key (DEK), which is stored in the database boot record for availability during recovery.

TDE protects data and log files, using AES and Triple Data Encryption Standard (3DES) encryption algorithms. Encryption of the database file is performed at the page level. The pages in an encrypted database are encrypted before they are written to disk and are decrypted when they're read into memory. TDE is now enabled by default on newly created Azure SQL databases.

Always Encrypted feature

With the Always Encrypted feature in Azure SQL you can encrypt data within client applications prior to storing it in Azure SQL Database. You can also enable delegation of on-premises database administration to third parties and maintain separation between those who own and can view the data and those who manage it but should not have access to it.

Cell-level or column-level encryption

With Azure SQL Database, you can apply symmetric encryption to a column of data by using Transact-SQL. This approach is called cell-level encryption or column-level encryption (CLE), because you can use it to encrypt specific columns or even specific cells of data with different encryption keys. Doing so gives you more granular encryption capability than TDE, which encrypts data in pages.

CLE has built-in functions that you can use to encrypt data by using either symmetric or asymmetric keys, the public key of a certificate, or a passphrase using 3DES.

Azure Cosmos DB database encryption

Azure Cosmos DB is Microsoft's globally distributed, multi-model database. User data that's stored in Azure Cosmos DB in non-volatile storage (solid-state drives) is encrypted by default. There are no controls to turn it on or off. Encryption at rest is implemented by using a number of security technologies, including secure key storage systems, encrypted networks, and cryptographic APIs. Encryption keys are managed by Microsoft and are rotated per Microsoft internal guidelines. Optionally, you can choose to add a second layer of encryption with keys you manage using the customer-managed keys or CMK feature.

At-rest encryption in Data Lake

Azure Data Lake is an enterprise-wide repository of every type of data collected in a single place prior to any formal definition of requirements or schema. Data Lake Store supports "on by default," transparent encryption of data at rest, which is set up during the creation of your account. By default, Azure Data Lake Store manages the keys for you, but you have the option to manage them yourself.

Three types of keys are used in encrypting and decrypting data: the Master Encryption Key (MEK), Data Encryption Key (DEK), and Block Encryption Key (BEK). The MEK is used to encrypt the DEK, which is stored on persistent media, and the BEK is derived from the DEK and the data block. If you are managing your own keys, you can rotate the MEK.

Encryption of data in transit

Azure offers many mechanisms for keeping data private as it moves from one location to another.

Data-link Layer encryption in Azure

Whenever Azure Customer traffic moves between datacenters-- outside physical boundaries not controlled by Microsoft (or on behalf of Microsoft)-- a data-link layer encryption method using the IEEE 802.1AE MAC Security Standards (also known as MACsec) is applied from point-to-point across the underlying network hardware. The packets are encrypted on the devices before being sent, preventing physical “man-in-the-middle” or snooping/wiretapping attacks. Because this technology is integrated on the network hardware itself, it provides line rate encryption on the network hardware with no measurable link latency increase. This MACsec encryption is on by default for all Azure traffic traveling within a region or between regions, and no action is required on customers’ part to enable.

TLS encryption in Azure

Microsoft gives customers the ability to use Transport Layer Security (TLS) protocol to protect data when it’s traveling between the cloud services and customers. Microsoft datacenters negotiate a TLS connection with client systems that connect to Azure services. TLS provides strong authentication, message privacy, and integrity (enabling detection of message tampering, interception, and forgery), interoperability, algorithm flexibility, and ease of deployment and use.

Perfect Forward Secrecy (PFS) protects connections between customers’ client systems and Microsoft cloud services by unique keys. Connections also use RSA-based 2,048-bit encryption key lengths. This combination makes it difficult for someone to intercept and access data that is in transit.

Azure Storage transactions

When you interact with Azure Storage through the Azure portal, all transactions take place over HTTPS. You can also use the Storage REST API over HTTPS to interact with Azure Storage. You can enforce the use of HTTPS when you call the REST APIs to access objects in storage accounts by enabling the secure transfer that's required for the storage account.

Shared Access Signatures (SAS), which can be used to delegate access to Azure Storage objects, include an option to specify that only the HTTPS protocol can be used when you use Shared Access Signatures. This approach ensures that anybody who sends links with SAS tokens uses the proper protocol.

SMB 3.0, which used to access Azure Files shares, supports encryption, and it's available in Windows Server 2012 R2, Windows 8, Windows 8.1, and Windows 10. It allows cross-region access and even access on the desktop.

Client-side encryption encrypts the data before it’s sent to your Azure Storage instance, so that it’s encrypted as it travels across the network.

SMB encryption over Azure virtual networks

By using SMB 3.0 in VMs that are running Windows Server 2012 or later, you can make data transfers secure by encrypting data in transit over Azure Virtual Networks. By encrypting data, you help protect against tampering and eavesdropping attacks. Administrators can enable SMB encryption for the entire server, or just specific shares.

By default, after SMB encryption is turned on for a share or server, only SMB 3.0 clients are allowed to access the encrypted shares.

In-transit encryption in VMs

Data in transit to, from, and between VMs that are running Windows can be encrypted in a number of ways, depending on the nature of the connection.

RDP sessions

You can connect and sign in to a VM by using the Remote Desktop Protocol (RDP) from a Windows client computer, or from a Mac with an RDP client installed. Data in transit over the network in RDP sessions can be protected by TLS.

You can also use Remote Desktop to connect to a Linux VM in Azure.

Secure access to Linux VMs with SSH

For remote management, you can use Secure Shell (SSH) to connect to Linux VMs running in Azure. SSH is an encrypted connection protocol that allows secure sign-ins over unsecured connections. It is the default connection protocol for Linux VMs hosted in Azure. By using SSH keys for authentication, you eliminate the need for passwords to sign in. SSH uses a public/private key pair (asymmetric encryption) for authentication.

Azure VPN encryption

You can connect to Azure through a virtual private network that creates a secure tunnel to protect the privacy of the data being sent across the network.

Azure VPN gateways

You can use an Azure VPN gateway to send encrypted traffic between your virtual network and your on-premises location across a public connection, or to send traffic between virtual networks.

Site-to-site VPNs use IPsec for transport encryption. Azure VPN gateways use a set of default proposals. You can configure Azure VPN gateways to use a custom IPsec/IKE policy with specific cryptographic algorithms and key strengths, rather than the Azure default policy sets.

Point-to-site VPNs

Point-to-site VPNs allow individual client computers access to an Azure virtual network. The Secure Socket Tunneling Protocol (SSTP) is used to create the VPN tunnel. It can traverse firewalls (the tunnel appears as an HTTPS connection). You can use your own internal public key infrastructure (PKI) root certificate authority (CA) for point-to-site connectivity.

You can configure a point-to-site VPN connection to a virtual network by using the Azure portal with certificate authentication or PowerShell.

To learn more about point-to-site VPN connections to Azure virtual networks, see:

Configure a point-to-site connection to a virtual network by using certification authentication: Azure portal

Configure a point-to-site connection to a virtual network by using certificate authentication: PowerShell

Site-to-site VPNs

You can use a site-to-site VPN gateway connection to connect your on-premises network to an Azure virtual network over an IPsec/IKE (IKEv1 or IKEv2) VPN tunnel. This type of connection requires an on-premises VPN device that has an external-facing public IP address assigned to it.

You can configure a site-to-site VPN connection to a virtual network by using the Azure portal, PowerShell, or Azure CLI.

For more information, see:

Create a site-to-site connection in the Azure portal

Create a site-to-site connection in PowerShell

Create a virtual network with a site-to-site VPN connection by using CLI

In-transit encryption in Data Lake

Data in transit (also known as data in motion) is also always encrypted in Data Lake Store. In addition to encrypting data prior to storing it in persistent media, the data is also always secured in transit by using HTTPS. HTTPS is the only protocol that is supported for the Data Lake Store REST interfaces.

To learn more about encryption of data in transit in Data Lake, see Encryption of data in Data Lake Store.

Key management with Key Vault

Without proper protection and management of the keys, encryption is rendered useless. Key Vault is the Microsoft-recommended solution for managing and controlling access to encryption keys used by cloud services. Permissions to access keys can be assigned to services or to users through Azure Active Directory accounts.

Key Vault relieves organizations of the need to configure, patch, and maintain hardware security modules (HSMs) and key management software. When you use Key Vault, you maintain control. Microsoft never sees your keys, and applications don’t have direct access to them. You can also import or generate keys in HSMs.

Next steps

- Azure security overview
- Azure network security overview
- Azure database security overview
- Azure virtual machines security overview

- Data encryption at rest
- Data security and encryption best practices

Additional resources

Documentation

Azure Data Encryption-at-Rest - Azure Security

This article provides an overview of Azure Data Encryption at-rest, the overall capabilities, and general considerations.

Data security and encryption best practices - Microsoft Azure

This article provides a set of best practices for data security and encryption using built in Azure capabilities.

Data encryption in Azure - Microsoft Azure Well-Architected Framework

Protect data in transit and at rest through encryption in Azure. Learn about Azure encryption features and standard encryption algorithms.

Data encryption models in Microsoft Azure

This article provides an overview of data encryption models In Microsoft Azure.

Double Encryption in Microsoft Azure

This article describes how Microsoft Azure provides double encryption for data at rest and data in transit.

Storage, data, and encryption in Azure - Microsoft Azure Well-Architected Framework

Secure data storage in Azure. Use identity-based storage access controls. Encrypt virtual disk files and data in transit. Enable platform encryption services.

Azure Managed HSM Overview - Azure Managed HSM

Azure Managed HSM is a cloud service that safeguards your cryptographic keys for cloud applications.

Overview of Key Management in Azure

This article provides an overview of Key Management in Azure.

Show 5 more

Training

Module

Encrypt public sector data in Azure - Training

Overview of the options available for managing encryption keys and encrypting data throughout its lifecycle, with special attention to public sector.

Certification

Microsoft Certified: Azure Data Fundamentals - Certifications

Azure Data Fundamentals validates foundational knowledge of core data concepts and how they are implemented using Microsoft Azure data services.

Applications and services

Article • 11/30/2022

Applications and the data associated with them act as the primary store of business value on a cloud platform. Applications can play a role in risks to the business because:

- **Business processes** are encapsulated and executed by applications and services need to be available and provided with high integrity.
- **Business data** is stored and processed by application workloads and requires high assurances of confidentiality, integrity, and availability.

Identify and classify business critical applications

Enterprise organizations typically have a large application portfolio, but not all applications have equal importance. Applications can be classified based on a criticality scale. For example, business-critical applications are designed to prevent financial losses, safety-critical are focused on costs associated with loss of human life. **Mission-critical applications** cover both aspects that can be impacted by unavailability or underperformance.

Criticality should be identified and classified, to direct investment of monitoring, time, and resources appropriately. You should also identify applications or systems with significant access — those which might grant control over other critical systems or data.

For more information, see Criticality scale in Cloud Adoption Framework.

Suggested actions

Identify and classify key organizational applications according to organizational impact.

Next steps

See these best practices related to PaaS applications.

Securing PaaS deployments

Secure communication paths between applications and the services. Make sure that there's a distinction between the endpoints exposed to the public internet and private ones. Also, the public endpoints are protected with web application firewall.

Network security

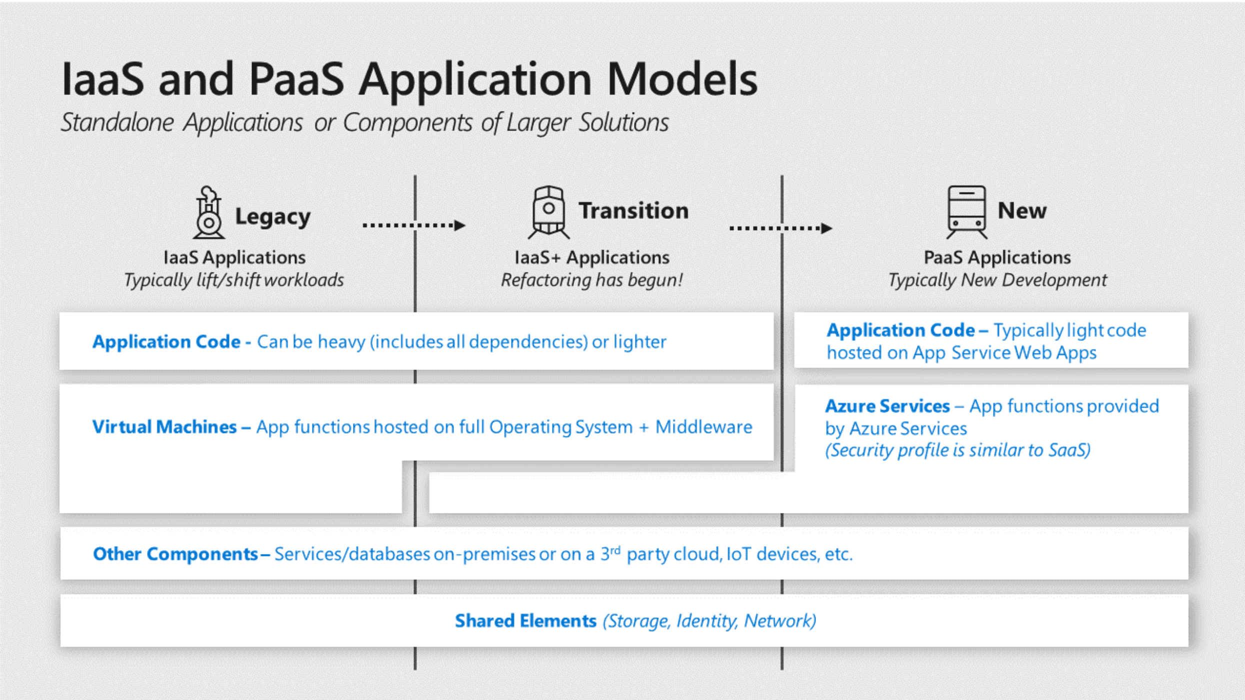
Application classification for security

Article • 11/30/2022

Azure can host both legacy and modern applications through Infrastructure as a Service (IaaS) virtual machines and Platform as a Service (PaaS). With legacy applications, you have the responsibility of securing all dependencies including OS, middleware, and other components. For PaaS applications, you don't need to manage and secure the underlying server OS. You are responsible for the application configuration.

This article describes the considerations for understanding the hosting models and the security responsibility of each, identifying critical applications.

Understand your responsibility as an owner



Securing an application requires security assurances for three aspects:

- **Application code**. The logic that defines the custom application that you write. Securing that code requires identifying and mitigating risks from the design and implementation of the application and assessing supply chain risk of included components.
- **Application services**. The cloud services that the application uses such as databases, identity providers, event hubs, IoT device management, and so on. Security for cloud services is a shared responsibility. The cloud provider ensures the security of the underlying service. The application owner is responsible for

security implications of the configuration and operation of the service instance(s) used by the application including any data stored and processed on the service.

- **Application hosting platform**. The computing environment where the application runs. This could take many forms with significant variations on who is responsible for security:

  - **Legacy applications**. typically require a full operating system (and any middleware) hosted on physical or virtualized hardware. This operating system and installed middleware/other components are operated and secured by the application owner or their infrastructure team(s). The security responsibility for the physical hardware and OS virtualization components (virtualization hosts, operating systems, and management services) varies:
    - **On-premises:** The application owner is responsible for maintenance and security.
    - **IaaS:** The cloud provider is responsible for the underlying infrastructure and the application owner's organization is responsible for the VM configuration, operating system, and any components installed on it.

  - **Modern applications** are hosted on PaaS environments such as an Azure application service. The underlying operating system is secured by the cloud provider. Application owners are responsible for the security of the application service configurations.

  - **Containers** are an application packaging mechanism in which applications are abstracted from the environment in which they run. The containerized applications can run on a container service by the cloud provider (modern applications) or on a server managed on premises or in IaaS.

Identify and classify applications

Identify applications that have a high potential impact and,or a high potential exposure to risk.

- **Business critical data**. Applications that process or store information must have assurance of confidentiality, integrity, and availability.
- **Regulated data**. Applications that handle monetary instruments and sensitive personal information regulated by standards such as the payment card industry (PCI), General Data Protection Regulation (GDPR), and Health Information Portability and Accountability Act (HIPAA).
- **Business critical availability**. Applications whose functionality is critical to the business mission, such as production lines generating revenue, devices or services

critical to life and safety, and other critical functions.

- **Significant Access**. Applications that have access to systems with a high impact through technical means such as:
  - Stored Credentials or keys/certificates that grant access to the data/service.
  - Permissions granted through access control lists or other methods.
- **High exposure to attacks**. Applications that are easily accessible to attackers such as web applications on the public internet. Legacy applications can also be higher exposure as attackers (and penetration testers) frequently target them because they know these legacy applications often have vulnerabilities that are difficult to fix.

Use Azure services for fundamental components

Developers should use services available from a cloud provider for well-established functions like databases, encryption, identity directory, and authentication, instead of building or adopting custom implementations, or third-party solutions that require integration with the cloud provider. These services provide better security, reliability, and efficiency because cloud providers operate and secure them with dedicated teams with deep expertise in those areas.

Using these services also frees your developer resources from reinventing the proverbial wheel so that they can focus development time on your unique requirements for your business. This practice should be followed to avoid risk during new application development and to reduce risk in existing applications either during the planned update cycle, or with a security-focused application update.

We recommend using cloud services from your cloud provider for identity, data protection, key management, and application configurations:

- **Identity:** User directories and other authentication functions are complex to develop and critically important to security assurances. Avoid custom authentication solutions. Instead choose native capabilities like Azure Active Directory (Azure AD), Azure AD B2B, Azure AD B2C, or third-party solutions to authenticate and grant permission to users, partners, customers, applications, services, and other entities. For more information, see Security with identity and access management (IAM) in Azure.

- **Data Protection:** Use established capabilities from cloud providers such as native encryption in cloud services to encrypt and protect data. If direct use of cryptography is required, use well-established cryptographic algorithms and not attempt to invent their own.

- **Key management:** Always authenticate with identity services rather than handling cryptographic key. For situations where you need to use keys, use a managed key store such as Azure Key Vault. This will make sure keys are handled safely in application code. Tools such as, CredScan can discover potentially exposed keys in your application code.
- **Application Configurations:** Inconsistent configurations for applications can create security risks. Application configuration information can be stored with the application itself or preferably using a dedicated configuration management system like Azure App Configuration or Azure Key Vault. App Configuration provides a service to centrally manage application settings and feature flags, which helps mitigate this risk. Don't store keys and secrets in application configuration.

For more information about using cloud services instead of custom implementations, reference Applications and services.

Use native capabilities

Use native security capabilities built into cloud services instead of adding external security components, such as data encryption, network traffic filtering, threat detection, and other functions.

Azure controls are maintained and supported by Microsoft. You don't have to invest in additional security tooling.

- List of Azure Services
- Native security capabilities of each service

Next steps

- Applications and services
- Application classification
- Application threat analysis
- Regulatory compliance

Application threat analysis

Article • 11/30/2022

Do a comprehensive analysis to identify threats, attacks, vulnerabilities, and counter measures. Having this information can protect the application and threats it might pose to the system. Start with simple questions to gain insight into potential risks. Then, progress to advanced techniques using threat modeling.

1- Gather information about the basic security controls

A threat modeling tool will produce a report of all threats identified. This report is typically uploaded into a tracking tool, or converted to work items that can be validated and addressed by the developers. As new features are added to the solution, the threat model should be updated and integrated into the code management process. If a security issue is found, there should be a process to triage issue severity and determine when and how to remediate (such as in the next release cycle, or a faster release).

Start by gathering information about each component of the application. The answers to these questions will identify gaps in basic protection and clarify the attack vectors.

Ask this question ...	To determine controls that ...
Are connections authenticated using Azure AD, TLS (with mutual authentication), or another modern security protocol approved by the security team? • Between users and the application • Between different application components and services	Prevent unauthorized access to the application component and data.
Are you limiting access to only those accounts that have the need to write or modify data in the application	Prevent unauthorized data tampering or alteration.
Is the application activity logged and fed into a Security Information and Event Management (SIEM) through Azure Monitor or a similar solution?	Detect and investigate attacks quickly.
Is critical data protected with encryption that has been approved by the security team?	Prevent unauthorized copying of data at rest.
Is inbound and outbound network traffic encrypted using TLS?	Prevent unauthorized copying of data in transit.

Ask this question ...	To determine controls that ...
Is the application protected against Distributed Denial of Service (DDoS) attacks using services such as Azure DDoS protection?	Detect attacks designed to overload the application so it can't be used.
Does the application store any logon credentials or keys to access other applications, databases, or services?	Identify whether an attack can use your application to attack other systems.
Do the application controls allow you to fulfill regulatory requirements?	Protect user's private data and avoid compliance fines.

Suggested actions

Assign tasks to the individual people who are responsible for a particular risk identified during threat modeling.

Learn more

Threat modeling

2- Evaluate the application design progressively

Analyze application components and connections and their relationships. Threat modeling is a crucial engineering exercise that includes defining security requirements, identifying and mitigating threats, and validating those mitigations. This technique can be used at any stage of application development or production, but it's most effective during the design stages of a new functionality.

Popular methodologies include:

- STRIDE:
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege

Microsoft Security Development Lifecycle uses STRIDE and provides a tool to assist with this process. This tool is available at no additional cost. For more information, see Microsoft Threat Modeling Tool .

- Open Web Application Security Project (OWASP) has documented a threat modeling approach for applications.

> Integrate threat modeling through automation using secure operations. Here are some resources:
>
> - Toolkit for Secure DevOps on Azure.
> - Guidance on DevOps pipeline security by OWASP.

3- Mitigate the identified threats

The threat modeling tool produces a report of all the threats identified. After a potential threat is identified, determine how it can be detected and the response to that attack. Define a process and timeline which minimizes exposure to any identified vulnerabilities in the workload, so that those vulnerabilities cannot be left unaddressed.

Use the *Defense-in-Depth* approach. This can help identify controls needed in the design to mitigate risk if a primary security control fails. Evaluate how likely it is for the primary control to fail. If it does, what is the extent of the potential organizational risk? Also, what is the effectiveness of the additional control (especially in cases that would cause the primary control to fail). Based on the evaluation apply Defense-in-Depth measures to address potential failures of security controls.

The principle of *least privilege* is one way of implementing Defense-in-Depth. It limits the damage that can be done by a single account. Grant least number of privileges to accounts that allows them to accomplish with the required permissions within a time period. This helps mitigate the damage of an attacker who gains access to the account to compromise security assurances.

There's often a disconnect between organizational leadership and technical teams regarding business requirements for critical workloads. This can create undesired outcomes and is especially sensitive when it pertains to information security. Routinely reviewing business critical workload requirements with executive sponsors to define requirements provides an opportunity to align expectations and ensure operational resource allocation to the initiative.

How are threats addressed once found?

Here are some best practices:

- Make sure the results are communicated to the interested teams.
- Prioritize the vulnerabilities and fix the most important in a timely manner.

- Upload the threat modeling report to a tracking tool. Create work items that can be validated and addressed by the developers. Cyber security teams can also use the report to determine attack vectors during a penetration test.
- As new features are added to the application, update the threat model report and integrate it into the code management process. Triage security issues into the next release cycle or a faster release, depending on the severity.

For information about mitigation strategies, see RapidAttack.

How long does it typically take to deploy a security fix into production?

If a security vulnerability is discovered, update the software with the fix as soon as possible. Have processes, tools, and approvals in place to roll out the fix quickly.

Learn more

Threat modeling

Next steps

- Applications and services
- Application classification
- Regulatory compliance

Securing PaaS deployments

Article • 04/02/2023 • 12 minutes to read

This article provides information that helps you:

- Understand the security advantages of hosting applications in the cloud
- Evaluate the security advantages of platform as a service (PaaS) versus other cloud service models
- Change your security focus from a network-centric to an identity-centric perimeter security approach
- Implement general PaaS security best practices recommendations

Develop secure applications on Azure is a general guide to the security questions and controls you should consider at each phase of the software development lifecycle when developing applications for the cloud.

Cloud security advantages

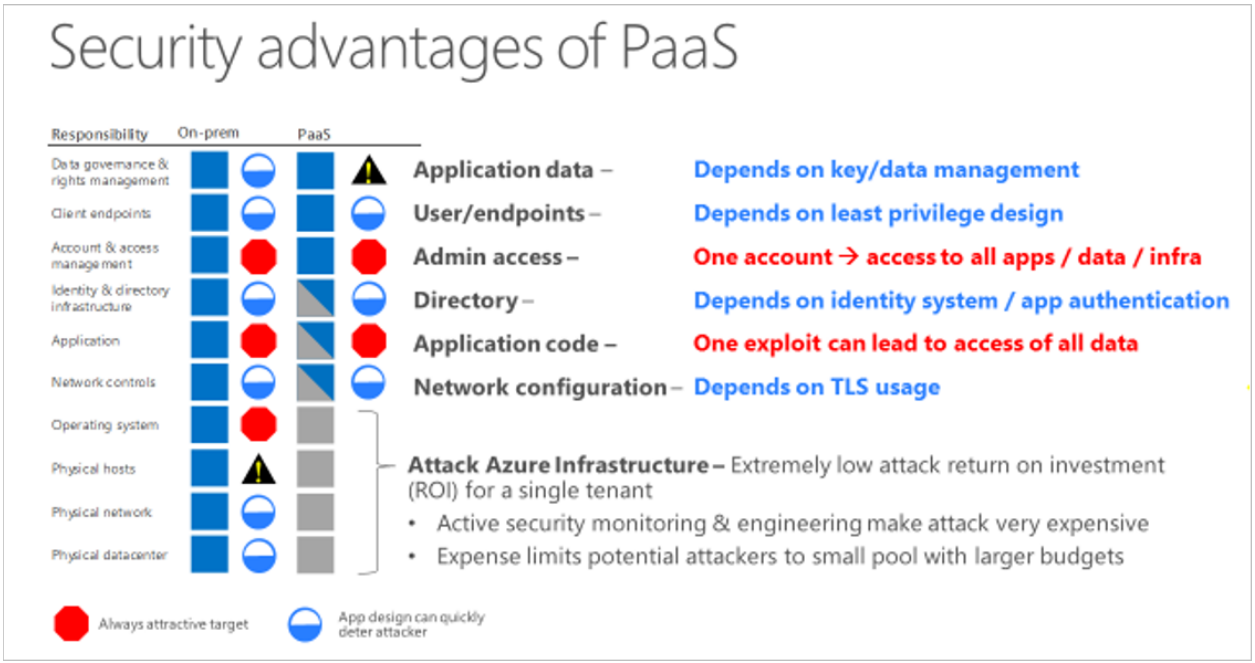
It's important to understand the division of responsibility between you and Microsoft. On-premises, you own the whole stack but as you move to the cloud some responsibilities transfer to Microsoft.

There are security advantages to being in the cloud. In an on-premises environment, organizations likely have unmet responsibilities and limited resources available to invest in security, which creates an environment where attackers are able to exploit vulnerabilities at all layers.

Organizations are able to improve their threat detection and response times by using a provider's cloud-based security capabilities and cloud intelligence. By shifting responsibilities to the cloud provider, organizations can get more security coverage, which enables them to reallocate security resources and budget to other business priorities.

Security advantages of a PaaS cloud service model

Let's look at the security advantages of an Azure PaaS deployment versus on-premises.



Starting at the bottom of the stack, the physical infrastructure, Microsoft mitigates common risks and responsibilities. Because the Microsoft cloud is continually monitored by Microsoft, it is hard to attack. It doesn't make sense for an attacker to pursue the Microsoft cloud as a target. Unless the attacker has lots of money and resources, the attacker is likely to move on to another target.

In the middle of the stack, there is no difference between a PaaS deployment and on-premises. At the application layer and the account and access management layer, you have similar risks. In the next steps section of this article, we will guide you to best practices for eliminating or minimizing these risks.

At the top of the stack, data governance and rights management, you take on one risk that can be mitigated by key management. (Key management is covered in best practices.) While key management is an additional responsibility, you have areas in a PaaS deployment that you no longer have to manage so you can shift resources to key management.

The Azure platform also provides you strong DDoS protection by using various network-based technologies. However, all types of network-based DDoS protection methods have their limits on a per-link and per-datacenter basis. To help avoid the impact of large DDoS attacks, you can take advantage of Azure's core cloud capability of enabling you to quickly and automatically scale out to defend against DDoS attacks. We'll go into more detail on how you can do this in the recommended practices articles.

Modernizing the Defender for Cloud's mindset

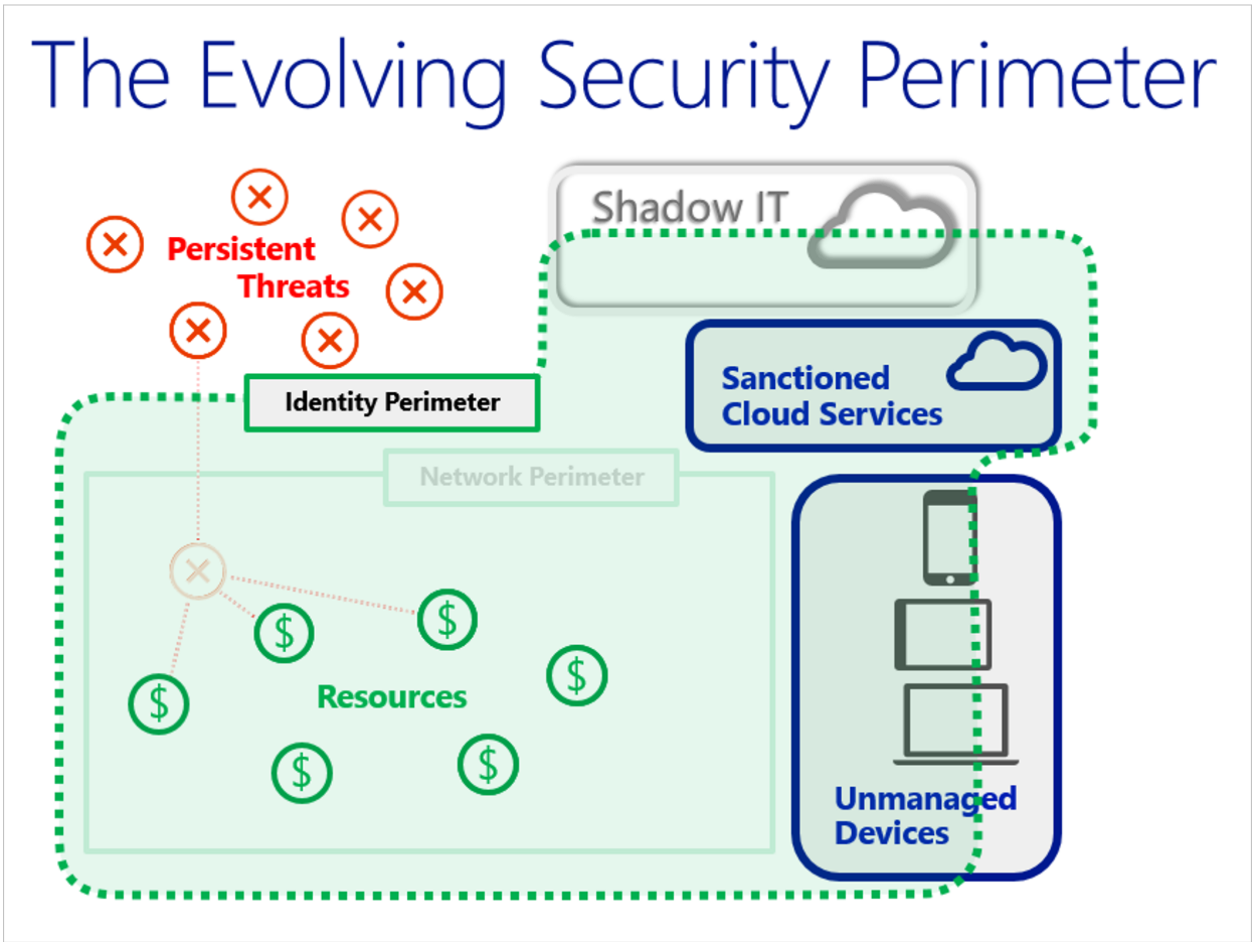
With PaaS deployments come a shift in your overall approach to security. You shift from needing to control everything yourself to sharing responsibility with Microsoft.

Another significant difference between PaaS and traditional on-premises deployments, is a new view of what defines the primary security perimeter. Historically, the primary on-premises security perimeter was your network and most on-premises security designs use the network as its primary security pivot. For PaaS deployments, you are better served by considering identity to be the primary security perimeter.

Adopt a policy of identity as the primary security perimeter

One of the five essential characteristics of cloud computing is broad network access, which makes network-centric thinking less relevant. The goal of much of cloud computing is to allow users to access resources regardless of location. For most users, their location is going to be somewhere on the Internet.

The following figure shows how the security perimeter has evolved from a network perimeter to an identity perimeter. Security becomes less about defending your network and more about defending your data, as well as managing the security of your apps and users. The key difference is that you want to push security closer to what's important to your company.



Initially, Azure PaaS services (for example, web roles and Azure SQL) provided little or no traditional network perimeter defenses. It was understood that the element's purpose was to be exposed to the Internet (web role) and that authentication provides the new perimeter (for example, BLOB or Azure SQL).

Modern security practices assume that the adversary has breached the network perimeter. Therefore, modern defense practices have moved to identity. Organizations must establish an identity-based security perimeter with strong authentication and authorization hygiene (best practices).

Principles and patterns for the network perimeter have been available for decades. In contrast, the industry has relatively less experience with using identity as the primary security perimeter. With that said, we have accumulated enough experience to provide some general recommendations that are proven in the field and apply to almost all PaaS services.

The following are best practices for managing the identity perimeter.

Best practice: Secure your keys and credentials to secure your PaaS deployment. **Detail**: Losing keys and credentials is a common problem. You can use a centralized solution where keys and secrets can be stored in hardware security modules (HSMs). Azure Key Vault safeguards your keys and secrets by encrypting authentication keys, storage account keys, data encryption keys, .pfx files, and passwords using keys that are protected by HSMs.

Best practice: Don't put credentials and other secrets in source code or GitHub. **Detail**: The only thing worse than losing your keys and credentials is having an unauthorized party gain access to them. Attackers can take advantage of bot technologies to find keys and secrets stored in code repositories such as GitHub. Do not put key and secrets in these public code repositories.

Best practice: Protect your VM management interfaces on hybrid PaaS and IaaS services by using a management interface that enables you to remote manage these VMs directly. **Detail**: Remote management protocols such as SSH, RDP, and PowerShell remoting can be used. In general, we recommend that you do not enable direct remote access to VMs from the internet.

If possible, use alternate approaches like using virtual private networks in an Azure virtual network. If alternative approaches are not available, ensure that you use complex passphrases and two-factor authentication (such as Azure AD Multi-Factor Authentication).

Best practice: Use strong authentication and authorization platforms. **Detail**: Use federated identities in Azure AD instead of custom user stores. When you use federated identities, you take advantage of a platform-based approach and you delegate the management of authorized identities to your partners. A federated identity approach is especially important when employees are terminated and that information needs to be reflected through multiple identity and authorization systems.

Use platform-supplied authentication and authorization mechanisms instead of custom code. The reason is that developing custom authentication code can be error prone. Most of your developers are not security experts and are unlikely to be aware of the subtleties and the latest developments in authentication and authorization. Commercial code (for example, from Microsoft) is often extensively security reviewed.

Use two-factor authentication. Two-factor authentication is the current standard for authentication and authorization because it avoids the security weaknesses inherent in username and password types of authentication. Access to both the Azure management (portal/remote PowerShell) interfaces and customer-facing services should be designed and configured to use Azure AD Multi-Factor Authentication.

Use standard authentication protocols, such as OAuth2 and Kerberos. These protocols have been extensively peer reviewed and are likely implemented as part of your platform libraries for authentication and authorization.

Use threat modeling during application design

The Microsoft Security Development Lifecycle specifies that teams should engage in a process called threat modeling during the design phase. To help facilitate this process, Microsoft has created the SDL Threat Modeling Tool. Modeling the application design and enumerating STRIDE threats across all trust boundaries can catch design errors early on.

The following table lists the STRIDE threats and gives some example mitigations that use Azure features. These mitigations won't work in every situation.

Threat	Security property	Potential Azure platform mitigations
Spoofing	Authentication	Require HTTPS connections.
Tampering	Integrity	Validate TLS/SSL certificates.
Repudiation	Non-repudiation	Enable Azure monitoring and diagnostics.

Threat	Security property	Potential Azure platform mitigations
Information disclosure	Confidentiality	Encrypt sensitive data at rest by using service certificates.
Denial of service	Availability	Monitor performance metrics for potential denial-of-service conditions. Implement connection filters.
Elevation of privilege	Authorization	Use Privileged Identity Management.

Develop on Azure App Service

Azure App Service is a PaaS offering that lets you create web and mobile apps for any platform or device and connect to data anywhere, in the cloud or on-premises. App Service includes the web and mobile capabilities that were previously delivered separately as Azure Websites and Azure Mobile Services. It also includes new capabilities for automating business processes and hosting cloud APIs. As a single integrated service, App Service brings a rich set of capabilities to web, mobile, and integration scenarios.

Following are best practices for using App Service.

Best practice: Authenticate through Azure Active Directory. **Detail**: App Service provides an OAuth 2.0 service for your identity provider. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, and mobile phones. Azure AD uses OAuth 2.0 to enable you to authorize access to mobile and web applications.

Best practice: Restrict access based on the need to know and least privilege security principles. **Detail**: Restricting access is imperative for organizations that want to enforce security policies for data access. You can use Azure RBAC to assign permissions to users, groups, and applications at a certain scope. To learn more about granting users access to applications, see Get started with access management.

Best practice: Protect your keys. **Detail**: Azure Key Vault helps safeguard cryptographic keys and secrets that cloud applications and services use. With Key Vault, you can encrypt keys and secrets (such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords) by using keys that are protected by hardware security modules (HSMs). For added assurance, you can import or generate keys in HSMs. See Azure Key Vault to learn more. You can also use Key Vault to manage your TLS certificates with auto-renewal.

Best practice: Restrict incoming source IP addresses. **Detail**: App Service Environment has a virtual network integration feature that helps you restrict incoming source IP addresses through network security groups. Virtual networks enable you to place Azure resources in a non-internet, routable network that you control access to. To learn more, see Integrate your app with an Azure virtual network.

Best practice: Monitor the security state of your App Service environments. **Detail**: Use Microsoft Defender for Cloud to monitor your App Service environments. When Defender for Cloud identifies potential security vulnerabilities, it creates recommendations that guide you through the process of configuring the needed controls.

Azure Cloud Services

Azure Cloud Services is an example of a PaaS. Like Azure App Service, this technology is designed to support applications that are scalable, reliable, and inexpensive to operate. In the same way that App Service is hosted on virtual machines (VMs), so too is Azure Cloud Services. However, you have more control over the VMs. You can install your own software on VMs that use Azure Cloud Services, and you can access them remotely.

Install a web application firewall

Web applications are increasingly targets of malicious attacks that exploit common known vulnerabilities. Common among these exploits are SQL injection attacks, cross site scripting attacks to name a few. Preventing such attacks in application code can be challenging and may require rigorous maintenance, patching and monitoring at many layers of the application topology. A centralized web application firewall helps make security management much simpler and gives better assurance to application administrators against threats or intrusions. A WAF solution can also react to a security threat faster by patching a known vulnerability at a central location versus securing each of individual web applications.

Web Application Firewall (WAF) provides centralized protection of your web applications from common exploits and vulnerabilities.

DDoS protection

Azure DDoS Protection Standard, combined with application-design best practices, provides enhanced DDoS mitigation features to provide more defense against DDoS attacks. You should enable Azure DDOS Protection Standard on any perimeter virtual network.

Monitor the performance of your applications

Monitoring is the act of collecting and analyzing data to determine the performance, health, and availability of your application. An effective monitoring strategy helps you understand the detailed operation of the components of your application. It helps you increase your uptime by notifying you of critical issues so that you can resolve them before they become problems. It also helps you detect anomalies that might be security related.

Use Azure Application Insights to monitor availability, performance, and usage of your application, whether it's hosted in the cloud or on-premises. By using Application Insights, you can quickly identify and diagnose errors in your application without waiting for a user to report them. With the information that you collect, you can make informed choices on your application's maintenance and improvements.

Application Insights has extensive tools for interacting with the data that it collects. Application Insights stores its data in a common repository. It can take advantage of shared functionality such as alerts, dashboards, and deep analysis with the Kusto query language.

Perform security penetration testing

Validating security defenses is as important as testing any other functionality. Make penetration testing a standard part of your build and deployment process. Schedule regular security tests and vulnerability scanning on deployed applications, and monitor for open ports, endpoints, and attacks.

Fuzz testing is a method for finding program failures (code errors) by supplying malformed input data to program interfaces (entry points) that parse and consume this data.

Next steps

In this article, we focused on security advantages of an Azure PaaS deployment and security best practices for cloud applications. Next, learn recommended practices for securing your PaaS web and mobile solutions using specific Azure services. We'll start with Azure App Service, Azure SQL Database and Azure Synapse Analytics, Azure

Storage, and Azure Cloud Services. As articles on recommended practices for other Azure services become available, links will be provided in the following list:

- Azure App Service
- Azure SQL Database and Azure Synapse Analytics
- Azure Storage
- Azure Cloud Services
- Azure Cache for Redis
- Azure Service Bus
- Web Application Firewall

See Develop secure applications on Azure for security questions and controls you should consider at each phase of the software development lifecycle when developing applications for the cloud.

See Azure security best practices and patterns for more security best practices to use when you're designing, deploying, and managing your cloud solutions by using Azure.

The following resources are available to provide more general information about Azure security and related Microsoft services:

- Microsoft Product Lifecycle - for consistent and predictable guidelines for support throughout the life of a product
- Microsoft Security Response Center - where Microsoft security vulnerabilities, including issues with Azure, can be reported or via email to secure@microsoft.com

Secure application configuration and dependencies

Article • 11/30/2022

Security of an application that is hosted in Azure is a shared responsibility between you as the application owner and Azure. For IaaS, you're responsible for configurations related to VM, operating system, and components installed on it. For PaaS, you're responsible for the security of the application service configurations and making sure that the dependencies used by the application are also secure.

Key points

- ✓ Don't store secrets in source code or configuration files. Instead, keep them in a secure store, such as Azure App Configuration or Azure Key Vault.
- ✓ Don't expose detailed error information when handling application exceptions.
- ✓ Don't expose platform-specific information.
- ✓ Store application configuration outside of the application code to update it separately and to have tighter access control.
- ✓ Restrict access to Azure resources that don't meet the security requirements.
- ✓ Validate the security of any open-source code added to your application.
- ✓ Update frameworks and libraries as part of the application lifecycle.

Configuration security

During the design phase, consider the way you store secrets and handle exceptions. Here are some points.

How is application configuration stored and how does the application access it?

Application configuration information can be stored with the application. However, that's not a recommended practice. Consider using a dedicated configuration management system such as Azure App Configuration or Azure Key Vault. That way, it can be updated independently of the application code.

Applications can include secrets like database connection strings, certificate keys, and so on. Don't store secrets in source code or configuration files. Instead, keep them in a secure store, such as Azure Key Vault. Identify secrets in code with static code scanning tools. Add the scanning process in your continuous integration (CI) pipeline.

For more information about secret management, reference Key and secret management.

Are errors and exceptions handled properly without exposing that information to users?

When handling application exceptions, make the application fail gracefully and log the error. Don't provide detailed information related to the failure, such as call stack, SQL queries, or out of range errors. This information can provide attackers with valuable information about the internals of the application.

Can configuration settings be changed or modified without rebuilding or redeploying the application?

Application code and configuration shouldn't share the same lifecycle to enable operational activities. These activities include those that change and update specific configurations without developer involvement or redeployment.

Is platform-specific information removed from server-client communication?

Don't reveal information about the application platform. Such information (for example, `X-Powered-By`, `X-ASPNET-VERSION`) can get exposed through HTTP banners, HTTP headers, error messages, and website footers. Malicious actors can use this information when mapping attack vectors of the application.

Suggested actions

Consider using Azure Front Door or API Management to remove platform-specific HTTP headers. Instead, use Azure CDN to separate the hosting platform from end users. Azure API Management offers transformation policies that allow you to modify HTTP headers and remove sensitive information.

Learn more

- Azure Front Door Rules Engine Actions
- API Management documentation

Are Azure policies used to control the configuration of the solution resources?

Use Azure Policy to deploy settings where applicable. Block resources that don't meet the proper security requirements defined during service enablement.

Dependencies, frameworks, and libraries

What are the frameworks and libraries used by the application?

Application frameworks are frequently updated and released by the vendor or communities. It's vital to track the frameworks and libraries used by the application including any resulting vulnerabilities they introduce. These frameworks and libraries include custom, OSS, third party, and others. Understand and manage the technologies the application uses, such as:

- .NET Core
- Spring
- Node.js

Automated solutions can help with this assessment.

Consider the following best practices:

- Validate the security of any open-source code added to your application. Free tools to help with this assessment include:
  - OWASP Dependency-Check
  - NPM audit
  - WhiteSource Bolt
  - GitHub Dependabot

  These tools find outdated components and update them to the latest versions.

- Maintain a list of frameworks and libraries as part of the application inventory. Also, keep track of versions in use. If vulnerabilities are published, this awareness helps to identify affected workloads.

- Update frameworks and libraries as part of the application lifecycle. Prioritize critical security patches.

Are the expiry dates of SSL/TLS certificates monitored and are processes in place to renew them?

Tracking expiry dates of SSL/TLS certificates and renewing them in due time is highly critical. Ideally, the process should be automated, although this often depends on the CA used for the certificate. If not automated, sufficient alerting should be applied to ensure expiry dates don't go unnoticed.

Learn more

- WhiteSource Bolt
- npm-audit
- OWASP Dependency-Check
- GitHub Dependabot

Referenced Azure services

- Azure Key Vault
- Azure CDN
- Azure Policy
- Azure Front Door
- Azure API Management

Next steps

- Applications and services
- Application classification
- Application threat analysis
- Regulatory compliance

Community resources

- OWASP Dependency-Check
- NPM audit

Secure deployment in Azure

Article • 11/30/2022

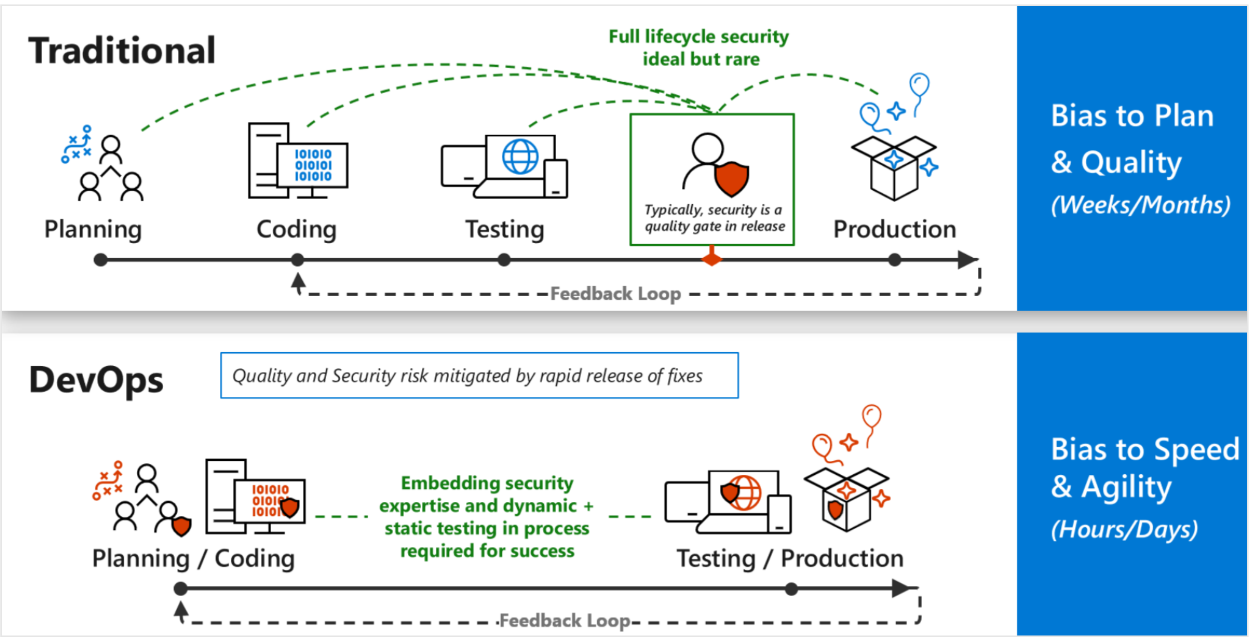
Have teams, processes, and tools that can quickly deploy security fixes? A *DevOps* or multidisciplinary approach is recommended. Multiple teams work together with efficient practices and tools. Essential DevOps practices include change management of the workload through continuous integration, continuous delivery (CI/CD).

Continuous integration (CI) is an automated process where code changes trigger the building and testing of the application. Continuous Delivery (CD) is an automated process to build, test, configure, and deploy the application from a build to production environment.

Those processes allow you to rapidly address the security concerns without waiting for a longer planning and testing cycle.

Building a DevOps process which includes a security discipline helps incorporate security concepts and enhancements earlier in the application development process. An organization's ability to rapidly address security and operational concerns increases through the combination of the Secure Development Lifecycle (SDL) and Operations Lifecycle related to application creation, maintenance, and updates.

Many traditional IT operating models aren't compatible with the cloud, and organizations must undergo operational and organizational transformation to deliver against enterprise migration targets. We recommend using a DevOps approach for both application and central teams.



Checklist

Have you adopted a secure DevOps approach to ensure security and feature enhancements can be quickly deployed?

- ✓ Establish a cross-functional DevOps platform team to build, manage, and maintain your workload.
- ✓ Involve the security team in the planning and design of the DevOps process to integrate preventive and detective controls for security risks.
- ✓ Clearly define CI/CD roles and permissions and minimize the number of people who have access to secure information or resources.
- ✓ Configure quality gate approvals in DevOps release process.
- ✓ Integrate scanning tools within CI/CD pipeline.
- ✓ No infrastructure changes, provisioning or configuring, should be done manually outside of IaC.

In this section

Follow these questions to assess the workload at a deeper level.

Assessment	Description
Do you clearly define CI/CD roles and permissions for this workload?	Define CI/CD permissions such that only users responsible for production releases can start the process and that only developers can access the source code.
Are any resources provisioned or operationally configured with user tools such as the Azure portal or via Azure CLI?	Always use Infrastructure as code (IaC) to make even the smallest of changes. This approach makes it easy to track code because the provisioned infrastructure is reproducible and reversible.
Can you roll back or forward code quickly through automated pipelines?	Automated deployment pipelines should allow for quick roll-forward and roll-back deployments to address critical bugs and code updates outside of the normal deployment lifecycle.

Azure security benchmark

The Azure Security Benchmark includes a collection of high-impact security recommendations. Use them to secure the services and processes you use to run the workload in Azure:

> The questions in this section are aligned to the Azure Security Benchmark controls.

Reference architecture

Here are some reference architectures related to building CI/CD pipelines:

- CI/CD for microservices architectures
- CI/CD for microservices on Kubernetes

Next step

We recommend monitoring activities that maintain the security posture. These activities can highlight, if the current security practices are effective or are there new requirements.

Security monitoring

Related link

> Go back to the main article: Security

Learn more

- Secure DevOps Kit for Azure
- Agile Principles in Practice
- Platform automation and DevOps
- PsRules for Azure

Governance considerations for secure deployment in Azure

Article • 11/30/2022

The automated continuous integration, continuous delivery (CI/CD) processes must have built-in governance that authorize and authenticate the identities to do the tasks within a defined scope.

Key points

- ✓ Clearly define CI/CD roles and permissions.
- ✓ Implement just-in-time privileged access management.
- ✓ Limit long-standing write access to production environments.
- ✓ Limit the scope of execution in the pipelines.
- ✓ Configure quality gate approvals in DevOps release process.

Minimize access

Minimize the number of people who have access to secure information or resources. This strategy will reduce the chance of a malicious actor gaining access or an authorized user inadvertently impacting a sensitive resource. Here are some considerations:

- Use the principle of least privilege when assigning roles and permissions. Only users responsible for production releases should start the process and only developers should access the source code.

  A pipeline should use one or more service principals. Ideally, they should be managed identity, and delivered by the platform and never directly defined within a pipeline. The identity should only have the Azure RBAC permissions necessary to do the task. All service principals should be bound to that pipeline and not shared across pipelines.

  How do you define CI/CD roles and permissions?

  Azure DevOps offers built-in roles that can be assigned to individual users of groups. If built-in roles are insufficient to define least privilege for a pipeline, consider creating custom Azure RBAC roles. Make sure those roles align with the action and the organization's teams and responsibilities.

To support security of your pipeline operations, you can add users to a built-in security group, set individual permissions for a user or group, or add users to pre-defined roles. You manage security for the following objects from Azure Pipelines in the web portal, either from the user or admin context.

For more information, see Get started with permissions, access, and security groups.

- For permissions, you grant or restrict permissions by setting the permission state to **Allow** or **Deny**, either for a security group or an individual user. For a role, you add a user or group to the role.

- Use separate pipeline identities between pre-production and production environments. If available, take advantage of pipeline features such as Environments to encapsulate last-mile authentication external to the executing pipeline.

- If the pipeline runs infrequently and has high privileges, consider removing standing permissions for that identity. Use just-in-time (JIT) role assignments, time-based, and approval-based role activation. This strategy will mitigate the risks of excessive, unnecessary, or misused access permissions on crucial resources. Azure AD Privileged Identity Management supports all those modes of activation.

- Review the organization's CI/CD pipeline and refine role assignment to create a clear delineation between development and production responsibilities.

Learn more

For more information about pipeline permission and security roles, reference Set different levels of pipeline permissions.

Execution scope

Where practical, limit the scope of execution in the pipelines.

Consider creating a multi-stage pipeline. Divide the work into discrete units and that can be isolated in a separate pipeline. Limit the identities only to the scope of the unit so that it has minimal privileges enough to do the action. For example, you can have two units, one to deploy and another that builds source code. Only allow the deploy unit to have access to the identity, not the build unit. If the build unit is compromised, it could start tampering with the infrastructure.

Gated approval process

Do you have release gate approvals configured in the DevOps release process?

Pull Requests and code reviews serve as the first line of approvals during development cycle. Before releasing an update to production, require a process that mandates security review and approval.

Make sure that you involve the security team in the planning, design, and DevOps process. This collaboration will help them implement security controls, auditing, and response processes.

Are branch policies used in source control management of this workload? How are they configured?

Establish branch policies that provide an extra level of control over the code that is committed to the repository. Lack of secure branch policy might allow poor, rogue or broken code to be checked-in and deployed. It's a common practice to deny pushes to the main branch if the change isn't approved. For example, you can require pull-request (PR) with code review before merging the changes by at least one reviewer, other than the change author.

Having multiple branches is recommended where each branch has a purpose and access level. For example, feature branches are created by developers and are open to push. Integration branch requires PR and code-review. Production branch requires another approval from the team lead before merging.

Suggested actions

- Configure quality gate approvals in DevOps release process.
- Follow the guidance in the linked articles to deploy and adopt branch strategy.

Learn more

- About branches and branch policies
- Adopt a Git branching strategy
- Release deployment control using gates

Next

Secure infrastructure deployments

Related links

> Go back to the main article: Secure deployment and testing in Azure

Infrastructure provisioning considerations in Azure

Article • 11/30/2022

Azure resources can be provisioned by code or user tools such as the Azure portal or via Azure CLI. It's not recommended that resources are provisioned or configured manually. Those methods are error prone and can lead to security gaps. Even the smallest of changes should be through code. The recommended approach is Infrastructure as code (IaC). It's easy to track because the provisioned infrastructure can be fully reproduced and reversed.

Key points

- ✓ No infrastructure changes should be done manually outside of IaC.
- ✓ Store keys and secrets outside of deployment pipeline in Azure Key Vault or in secure store for the pipeline.
- ✓ Incorporate security fixes and patching to the operating system and all parts of the codebase, including dependencies (preinstalled tools, frameworks, and libraries).

Infrastructure as code (IaC)

Make all operational changes and modifications through IaC. This is a key DevOps practice, and it's often used with continuous delivery. IaC manages the infrastructure - such as networks, virtual machines, and others - with a descriptive model, using a versioning system that is similar to what is used for source code. IaC model generates the same environment every time it's applied. Common examples of IaC are Azure Resource Manager, Bicep or Terraform.

IaC reduces configuration effort and automates full environment deployment (production and pre-production). Also, IaC allows you to develop and release changes faster. All those factors enhance the security of the workload.

For detailed information about IaC, see What is Infrastructure as code (IaC).

Pipeline secret management

How are credentials, certificates, and other secrets used in the operations for the workload managed during deployment?

Store keys and secrets outside of deployment pipeline in a managed key store, such as Azure Key Vault. Or, in a secure store for the pipeline. When deploying application infrastructure with Azure Resource Manager, Bicep or Terraform, the process might generate credentials and keys. Store them in a managed key store and make sure the deployed resources reference the store. Do not hard-code credentials.

Secret scanning tools like GitHub Secret Scanner can be used to scan for existing hard-coded credentials. Add the scanning process in your continuous integration (CI) pipeline to prevent new hard-coded credentials from being added.

Build environments

Does the organization apply security controls (IP firewall restrictions, update management) to self-hosted build agents for this workload?

Custom build agents add management complexity and can become an attack vector. Build machine credentials must be stored securely and the file system needs to be cleaned of any temporary build artifacts regularly. Network isolation can be achieved by only allowing outgoing traffic from the build agent, because it's using the pull model of communication with Azure DevOps.

As part of the operational lifecycle, incorporate security fixes and patching to the operating system and all parts of the codebase, including dependencies (preinstalled tools, frameworks, and libraries).

Apply security controls to self-hosted build agents in the same manner as with other Azure IaaS VMs. These should be minimalistic environments as a way to reduce the attack surface.

Learn more

- Azure Pipelines agents
- I'm running a firewall and my code is in Azure Repos. What URLs does the agent need to communicate with?

Next step

Secure code deployments

Related links

> Go back to the main article: [Secure deployment and testing in Azure]

Code deployments

Article • 11/30/2022

The automated build and release pipelines should update a workload to a new version seamlessly without breaking dependencies. Augment the automation with processes that allow high priority fixes to get deployed quickly.

Organizations should leverage existing guidance and automation when securing applications in the cloud, rather than starting from zero. Using resources and lessons learned by external organizations that are early adopters of these models can accelerate the improvement of an organizations security posture with less expenditure of effort and resources.

Key points

- ✓ Involve the security team in the planning and design of the DevOps process to integrate preventive and detective controls for security risks.
- ✓ Design automated deployment pipelines that allow for quick roll-forward and rollback deployments to address critical bugs and code updates outside of the normal deployment lifecycle.
- ✓ Integrate code scanning tools within CI/CD pipeline.

Rollback and roll-forward

If something goes wrong, the pipeline should roll back to a previous working version. N-1 and N+1 refer to rollback and roll-forward versions. Automated deployment pipelines should allow for quick roll-forward and rollback deployments to address critical bugs and code updates outside of the normal deployment lifecycle.

Can N-1 or N+1 versions be deployed via automated pipelines where N is current deployment version in production?

Because security updates are a high priority, design a pipeline that supports regular updates and critical security fixes.

A release is typically associated with approval processes with multiple sign-offs, quality gates, and so on. If the workload deployment is small with minimal approvals, you can usually use the same process and pipeline to release a security fix.

An approval process that is complex and takes a significant amount of time can delay a fix. Consider building an emergency process to accelerate high priority fixes. The process might be business and, or communication process between teams. Another way is to build a pipeline that might not include all the gated approvals, but should be able to push out the fix quickly. The pipeline should allow for quick roll-forward and rollback deployments that address security fixes, critical bugs, and code updates outside of the regular deployment life cycle.

> ⓘ **Important**
>
> Deploying a security fix is a priority, but it shouldn't be at the cost of introducing a regression or bug. When designing an emergency pipeline, carefully consider which automated tests can be bypassed. Evaluate the value of each test against the execution time. For example, unit tests usually complete quickly. Integration or end-to-end tests can run for a long time.

Involve the security team in the planning and design of the DevOps process. Your automated pipeline design should have the flexibility to support both regular and emergency deployments. This is important to support the rapid and responsible application of both security fixes and other urgent, important fixes.

Suggested action

Implement an automated deployment process with support for rollback scenarios via Azure App Services deployment slots.

Learn more

Set up staging environments in Azure App Service

Credential scanning

Credentials, keys, and certificates grant access to the data or service used by the workload. Storing credentials in code poses a significant security risk. Ensure that static code scanning tools are an integrated part of the continuous integration (CI) process.

Are code scanning tools an integrated part of the continuous integration (CI) process for this workload?

To prevent credentials from being stored in the source code or configuration files, integrate code scanning tools within the CI/CD pipeline:

- During design time, use code analyzers to prevent credentials from getting pushed to the source code repository. For example, .NET Compiler Platform (Roslyn)

Analyzers inspect your C# or Visual Basic code.
- During the build process, use pipeline add-ons to catch credentials in the source code. Some options include GitHub Advanced Security and OWASP source code analysis tools.
- Scan all dependencies, such as third-party libraries and framework components, as part of the CI process. Investigate vulnerable components that are flagged by the tool. Combine this task with other code scanning tasks that inspect code churn, test results, and coverage.
- Use a combination of dynamic application security testing (DAST) and static application security testing (SAST). DAST tests the application while its in use. SAST scans the source code and detects vulnerabilities based on its design or implementation. Some technology options are provided by OWASP. For more information, see SAST Tools and Vulnerability Scanning Tools.
- Use scanning tools that are specialized in technologies used by the workload. For example, if the workload is containerized, run container-aware scanning tools to detect risks in the container registry, before use, and during use.

Suggested actions

Incorporate Secure DevOps on Azure toolkit and the guidance published by the Organization for Web App Security Project (OWASP), or an equivalent guiding organization.

Learn more

- Follow DevOps security guidance
- Getting started with Credential Scanner (CredScan)

Community links

- OWASP source code analysis tools
- GitHub Advanced Security
- Vulnerability Scanning Tools

> Go back to the main article: Secure deployment and testing in Azure

Security monitoring and remediation in Azure

Article • 11/30/2022

Regularly monitor resources to maintain the security posture and detect vulnerabilities. Detection can take the form of reacting to an alert of suspicious activity or proactively hunting for anomalous events in the enterprise activity logs. vigilantly responding to anomalies and alerts to prevent security assurance decay, and designing for defense in depth and least privilege strategies.

Checklist

How are you monitoring security-related events in this workload?

- ✓ Use native tools in Azure to monitor the workload resources and the infrastructure in which it runs.
- ✓ Consider investing in a Security Operations Center (SOC), or SecOps team and incident response plan.
- ✓ Monitor traffic, access requests, and application communication between segments.
- ✓ Discover and remediate common risks to improve secure score in Microsoft Defender for Cloud.
- ✓ Use an industry standard benchmark to evaluate the security posture by learning from external organizations.
- ✓ Send logs and alerts to a central security log management for analysis.
- ✓ Perform regular internal and external compliance audits, including regulatory compliance attestations.
- ✓ Regularly test your security design and implementation using test cases based on real-world attacks.

Azure security benchmark

The Azure Security Benchmark includes a collection of high-impact security recommendations. Use them to secure the services and processes you use to run the workload in Azure:

> The questions in this section are aligned to these controls:
>
> - Azure Security Benchmarks Logging and threat detection.

> - Azure Security Benchmarks Incident response.
> - Posture and Vulnerability Management

Reference architecture

- Hybrid Security Monitoring using Microsoft Defender for Cloud and Microsoft Sentinel

  This reference architecture illustrates how to use Microsoft Defender for Cloud and Microsoft Sentinel to monitor the security configuration and telemetry of on-premises and Azure operating system workloads.

- Azure security solutions for AWS

  This article provides AWS identity architects, administrators, and security analysts with immediate insights and detailed guidance for deploying several Microsoft security solutions.

Next step

We recommend applying as many best practices as early as possible, and then working to retrofit any gaps over time as you mature your security program.

Related link

> Go back to the main article: Security

Azure security monitoring tools

Article • 03/24/2023

The *leverage native control* security principle tells us to use native controls built over third-party solutions. Native reduce the effort required to integrate external security tooling and update those integrations over time.

Azure provides several monitoring tools that observe the operations and detect anomalous behavior. These tools can detect threats at different levels and report issues. Addressing the issues early in the operational lifecycle will strengthen your overall security posture.

Tools

Service	Use case
Microsoft Defender for Cloud	Strengthens the security posture of your data centers, and provides advanced threat protection across your workloads in the cloud (whether they're in Azure or not) and on-premises. Get a unified view into the infrastructure and resources provisioned for the workload.
Microsoft Sentinel	Use the native security information event management (SIEM) and security orchestration automated response (SOAR) solution on Azure. Receive intelligent security analytics and threat intelligence across the enterprise.
Azure DDoS Protection	Defend against distributed denial of service (DDoS) attacks.
Azure Rights Management (RMS)	Protect files and emails across multiple devices.
Microsoft Purview Information Protection	Secure email, documents, and sensitive data that you share outside your company.
Azure Governance Visualizer	Gain granular insight into policies, Azure role-based access control (Azure RBAC), Azure Blueprints, subscriptions, and more.
PSRule for Azure	Scans Azure Infrastructure as Code (IaC) artifacts for issues across Azure Well-Architected pillars.

Microsoft Defender for Cloud

Enable Microsoft Defender for Cloud at the subscription level to monitor all resource provisioned in that scope. At no additional cost, it provides continuous observability into resources, reports issues, and recommends fixes. By regularly reviewing and fixing issues, you can improve the security posture, detect threats early, prevent breaches.

Beyond just observability, Defender for Cloud offers an advanced mode through its integration with **Microsoft Defender for Cloud**. When these plans are enabled, built-in policies, custom policies, and initiatives protect resources and block malicious actors. You can also monitor compliance with regulatory standards - such as NIST, Azure CIS, Azure Security Benchmark. For pricing details, see Defender for Cloud pricing.

Microsoft Sentinel

Your organization might run workloads on multiple cloud platforms, and, or across cloud and on-premises, or managed by various teams within the organization. Having a centralized view of all data is recommended. To get that view you need security information event management (SIEM) and security orchestration automated response (SOAR) solutions. These solutions connect to all security sources, monitor them, and analyze the correlated data.

Microsoft Sentinel and is a native control that combines SIEM and SOAR capabilities. It analyzes events and logs from various connected sources. Based on the data sources and their alerts, Sentinel creates incidents, performs threat analysis for early detection. Through intelligent analytics and queries, you can be proactive with hunting activities. In case of incidents, you can automate workflows. Also, with workbook templates you can quickly gain insights through visualization.

Azure DDoS Protection

A Distributed Denial of Service (DDoS) attack attempts to exhaust an application's resources, making the application unavailable to legitimate users. DDoS attacks can be targeted at any endpoint that is publicly reachable through the internet.

Every property in Azure is protected by Azure's infrastructure DDoS (Basic) Protection at no additional cost. The scale and capacity of the globally deployed Azure network provides defense against common network-layer attacks through always-on traffic monitoring and real-time mitigation. DDoS Infrastructure Protection requires no user configuration or application changes. DDoS Infrastructure Protection helps protect all Azure services, including PaaS services like Azure DNS.

Azure DDoS Protection provides enhanced DDoS mitigation features to defend against DDoS attacks. It's automatically tuned to help protect your specific Azure resources in a virtual network. Protection is simple to enable on any new or existing virtual network, and it requires no application or resource changes. It has several advantages over the basic service, including logging, alerting, telemetry, SLA guarantee, and cost protection.

Azure DDoS Network Protection is designed for services that are deployed in a virtual network. For other services, the default DDoS Infrastructure Protection service applies. To learn more about supported architectures, see DDoS Protection reference architectures.

Azure Rights Management (RMS)

Your business may encounter challenges with protecting documents and emails. For example, file protection, collaboration, and sharing may be issues. You also might be experiencing problems regarding platform support or infrastructure.

Azure Rights Management (RMS) is a cloud-based protection service. RMS uses encryption, identity, and authorization policies to help secure files and emails across devices, including phones, tablets, and PCs.

To learn more about how RMS can address these issues, see Business problems solved by Azure Rights Management.

Microsoft Purview Information Protection

The *data classification* process categorizes data by sensitivity and business impact in order to identify risks. When data is classified, you can manage it in ways that protect sensitive or important data from theft or loss.

With proper *file protection*, you can analyze data flows to gain insight into your business, detect risky behaviors and take corrective measures, track access to documents, and more. The protection technology in AIP uses encryption, identity, and authorization policies. Protection stays with the documents and emails, independently of the location, regardless of whether they're inside or outside your organization, networks, file servers, and applications

Azure Information Protection (AIP) is part of Microsoft Purview Information Protection solution, and extends the labeling and classification functionality provided by Microsoft 365. For more information, see this article about classification.

Azure Governance Visualizer

Azure Governance Visualizer is a PowerShell script that iterates through an Azure tenant's management group hierarchy down to the subscription level. You can run the script either for your Tenant Root Group or any other Management Group. It captures data from the most relevant Azure governance capabilities such as Azure Policy, Azure role-based access control (Azure RBAC), and Azure Blueprints. From the collected data, the visualizer shows your hierarchy map, creates a tenant summary, and builds granular scope insights about your management groups and subscriptions.

The visualizer provides a holistic overview of your technical Azure Governance implementation by connecting the dots.

PSRule for Azure

PSRule for Azure is a set of tests and documentation to help you configure Azure solutions. These tests allow you to check your Azure Template or Bicep Infrastructure as Code (IaC) before deployment to Azure. PSRule for Azure includes tests that check how IaC is written and how Azure resources are configured.

Next

Monitor workload resources in Microsoft Defender for Cloud

Related links

For information on the Microsoft Defender for Cloud tools, see Strengthen security posture.

For frequently asked questions on Microsoft Defender for Cloud, see FAQ - General Questions.

For information on the Microsoft Sentinel tools that will help to meet these requirements, see What is Microsoft Sentinel?

For types of DDoS attacks that DDoS Protection mitigates as well as more features, see Azure DDoS Protection overview.

Monitor Azure resources in Microsoft Defender for Cloud

Article • 04/19/2023

Most cloud architecture have compute, networking, data, and identity components and each require different monitoring mechanisms. Even Azure services have individual monitoring needs. For instance, to monitor Azure Functions you want to enable Azure Application Insights.

Microsoft Defender for Cloud has many plans that monitor the security posture of machines, networks, storage and data services, and applications to discover potential security issues. Common issues include internet connected VMs, or missing security updates, missing endpoint protection or encryption, deviations from baseline security configurations, missing Web Application Firewall (WAF), and more.

Key points

- ✓ Enable Microsoft Defender for Cloud as a defense-in-depth measure. Use resource-specific Defender for Cloud features such as Microsoft Defender for servers, Microsoft Defender for Endpoint, Microsoft Defender for Storage.
- ✓ Observe container hygiene through container aware tools and regular scanning.
- ✓ Review all network flow logs through network watcher. See diagnostic logs in Microsoft Defender for Cloud.
- ✓ Integrate all logs in a central SIEM solution to analyze and detect suspicious behavior.
- ✓ Monitor identity-related risk events in Azure AD reporting and Azure Active Directory Identity Protection.

General best practices

Identifying common security activities will significantly reduce the overall risk.

- Monitor suspicious activities from administrative accounts.
- Monitor the location from where Azure resources are being managed.
- Monitor attempts to access deactivated credentials.
- Use automated tools to monitor network resource configurations and detect changes.

For more information, see Azure security baseline for Azure Monitor.

IaaS and PaaS security

In an IaaS model, you can host the workload on Azure infrastructure. Azure provides security assurances that maintain isolation and timely security updates to the infrastructure. For greater control, you host the entire IaaS solution on-premises or in a hosted data center and are responsible for security. You must implement security on the host, virtual machine, network, and storage. For instance if you have your own VNet, consider enabling Azure Private Link over Azure Monitor so you can access this over a private endpoint.

In PaaS, you have shared responsibility with Azure in protecting the data.

Virtual machines

If you're running your own Windows and Linux virtual machines, use Microsoft Defender for Cloud. Take advantage of the free services to check for missing OS patches, security misconfiguration, and basic network security. Enabling Microsoft Defender for Cloud is highly recommended because you get features that provide adaptive application controls, file integrity monitoring (FIM), and others.

For example, a common risk is the virtual machines don't have vulnerability scanning solutions that check for threats. Microsoft Defender for Cloud reports those machines. You can remediate in Microsoft Defender for Cloud by deploying a scanning solution. You can use the built-in vulnerability scanner for virtual machines. You don't need a license. Instead, you can bring your license for supported partner solutions.

> **Note**
>
> Vulnerability assessments are also available for container images, and SQL servers.

Attackers constantly scan public cloud IP ranges for open management ports, which can lead to attacks such as common passwords and known unpatched vulnerabilities. JIT (Just In Time) access allows you to lock down the inbound traffic to the virtual machines while providing easy access to connect to machines when needed. Defender for Cloud identifies which machines should have JIT applied.

With Microsoft Defender for Cloud, you also get Microsoft Defender for Endpoint. This provides investigative tools Endpoint Detection and Response (EDR) that helps in threat detection and analysis.

Microsoft Defender for servers also watches the network to and from virtual machines. If you are using network security groups to control access to the virtual machines and the rules are overpermissive, Defender for Cloud will flag them. Adaptive network hardening provides recommendations to further harden the NSG rules.

For a full list of features, see Feature coverage for machines.

Remove direct internet connectivity

Make sure policies and processes require restricting and monitoring direct internet connectivity by virtual machines.

For Azure, you can enforce policies by:

- **Enterprise-wide prevention:** Prevent inadvertent exposure by following the permissions and roles described in the reference model.
    - Ensures that network traffic is routed through approved egress points by default.
    - Exceptions (such as adding a public IP address to a resource) must go through a centralized group that evaluates exception requests and makes sure appropriate controls are applied.
- **Identify and remediate** exposed virtual machines by using the Microsoft Defender for Cloud network visualization to quickly identify internet exposed resources.
- **Restrict management ports** (RDP, SSH) using Just in Time access in Microsoft Defender for Cloud.

One way of managing VMs in the virtual network is by using Azure Bastion. This service allows you to log into VMs in the virtual network through SSH or remote desktop protocol (RDP) without exposing the VMs directly to the internet. To see a reference architecture that uses Bastion, see Network DMZ between Azure and an on-premises datacenter.

Containers

Containerized workloads have an extra layer of abstraction and orchestration. That complexity requires specific security measures that protect against common container attacks such as supply chain attacks.

- Use container registries that are validated for security. Images in public registries might contain malware or unwanted applications that activate when the container is running. Build a process for developers to request and rapidly get security validation of new containers and images. The process should validate against your security standards. This includes applying security updates, scanning for unwanted code such as backdoors and illicit crypto coin miners, scanning for security vulnerabilities, and application of secure development practices.

A popular process pattern is the quarantine pattern. This pattern allows you to get your images on a dedicated container registry and subject them to security or compliance scrutiny applicable for your organization. After it's validated, they can then be released from quarantine and promoted to being available.

Microsoft Defender for Cloud identifies unmanaged containers hosted on IaaS Linux VMs, or other Linux machines running Docker containers.

- Make sure you use images from authorized registries. You can enforce this restriction through Azure Policy. For example, for an Azure Kubernetes Service (AKS) cluster, have policies that restrict the cluster to only pull images from Azure Container Registry (ACR) that is deployed as part of the architecture.

> **Tip**
>
> Here are the resources for the preceding example:
>
> **GitHub: Azure Kubernetes Service (AKS) Secure Baseline Reference Implementation**.
>
> The design considerations are described in **Baseline architecture for an AKS cluster**.

- Regularly scan containers for known risks in the container registry, before use, and during use.

- Use security monitoring tools that are container aware to monitor for anomalous behavior and enable investigation of incidents.

  Microsoft Defender for container registries are designed to protect AKS clusters, container hosts (virtual machines running Docker), and ACR registries. When enabled, the images that are pulled or pushed to registries are subject to vulnerability scans.

For more information, see these articles:

- Container security in Defender for Cloud

Network

How do you monitor and diagnose conditions of the network?

As an initial step, enable and review all logs (including raw traffic) from your network devices.

- Security group logs – flow logs and diagnostic logs
- Azure Network Watcher

Take advantage of the packet capture feature to set alerts and gain access to real-time performance information at the packet level.

Packet capture tracks traffic in and out of virtual machines. It gives you the capability to run proactive captures based on defined network anomalies including information about network intrusions.

For an example, see Scenario: Get alerts when VM is sending you more TCP segments than usual.

Then, focus on observability of specific services by reviewing the diagnostic logs. For example, for Azure Application Gateway with integrated WAF, see Web application firewall logs. Microsoft Defender for Cloud analyzes diagnostic logs on virtual networks, gateways, network security groups and determines if the controls are secure enough. For example:

- Is your virtual machine exposed to public internet. If so, do you have tight rules on network security groups to protect the machine?
- Are the network security groups (NSG) and rules that control access to the virtual machines overly permissive?
- Are the storage accounts receiving traffic over secure connections?

Follow the recommendations provided by Defender for Cloud. For more information, see Networking recommendations. Use Azure Firewall logs and metrics for observability into operational and audit logs.

Integrate all logs into a security information and event management (SIEM) service, such as Microsoft Sentinel. The SIEM solutions support ingestion of large amounts of information and can analyze large datasets quickly. Based on those insights, you can:

- Set alerts or block traffic crossing segmentation boundaries.
- Identify anomalies.
- Tune the intake to significantly reduce the false positive alerts.

Identity

Monitor identity-related risk events using adaptive machine learning algorithms, heuristics quickly before the attacker can gain deeper access into the system.

Review identity risks

Most security incidents take place after an attacker initially gains access using a stolen identity. Even if the identity has low privileges, the attacker can use it to traverse laterally and gain access to more privileged identities. This way the attacker can control access to the target data or systems.

Does the organization actively monitor identity-related risk events related to potentially compromised identities?

Monitor identity-related risk events on potentially compromised identities and remediate those risks. Review the reported risk events in these ways:

- Azure AD reporting. For information, see users at risk security report and the risky sign-ins security report.
- Use the reporting capabilities of Azure Active Directory Identity Protection.
- Use the Identity Protection risk events API to get programmatic access to security detections by using Microsoft Graph. See riskDetection and riskyUser APIs.

Azure AD uses adaptive machine learning algorithms, heuristics, and known compromised credentials (username/password pairs) to detect suspicious actions that are related to your user accounts. These username/password pairs come from monitoring public and dark web and by working with security researchers, law enforcement, security teams at Microsoft, and others.

Remediate risks by manually addressing each reported account or by setting up a user risk policy to require a password change for high risk events.

Regularly review critical access

Regularly review roles that are assigned privileges with a business-critical impact.

Set up a recurring review pattern to ensure that accounts are removed from permissions as roles change. You can conduct the review manually or through an automated process by using tools such as Azure AD access reviews.

Discover & replace insecure protocols

Discover and disable the use of legacy insecure protocols SMBv1, LM/NTLMv1, wDigest, Unsigned LDAP Binds, and Weak ciphers in Kerberos.

Applications should use the SHA-2 family of hash algorithms (SHA-256, SHA-384, SHA-512). Use of weaker algorithms, like SHA-1 and MD5, should be avoided.

Authentication protocols are a critical foundation of nearly all security assurances. These older versions can be exploited by attackers with access to your network and are often used extensively on legacy systems on Infrastructure as a Service (IaaS).

Here are ways to reduce your risk:

- Discover protocol usage by reviewing logs with Microsoft Sentinel's Insecure Protocol Dashboard or third-party tools.
- Restrict or Disable use of these protocols by following guidance for SMB, NTLM, WDigest.
- Use only secure hash algorithms (SHA-2 family).

We recommend implementing changes using pilot or other testing methods to mitigate risk of operational interruption.

Learn more

For more information about hash algorithms, see Hash and Signature Algorithms.

Connected tenants

Does your security team have visibility into all existing subscriptions and cloud environments? How do they discover new ones?

Make sure the security team is aware of all enrollments and associated subscriptions connected to your existing environment through ExpressRoute or Site-Site VPN. Monitor them as part of the overall enterprise.

Assess if organizational policies and applicable regulatory requirements are followed for the connected tenants. This applies to all Azure environments that connect to your production environment network.

The organizations' cloud infrastructure should be well documented, with security team access to all resources required for monitoring and insight. Conduct frequent scans of the cloud-connected assets to ensure no additional subscriptions or tenants have been added outside of organizational controls. Regularly review Microsoft guidance to ensure security team access best practices are consulted and followed.

Suggested actions

Ensure all Azure environments that connect to your production environment and network apply your organization's policy, and IT governance controls for security.

You can discover existing connected tenants using a tool provided by Microsoft. Guidance on permissions you may assign to security is in the Assign privileges for managing the environment section.

CI/CD pipelines

DevOps practices are for change management of the workload through continuous integration, continuous delivery (CI/CD). Make sure you add security validation in the pipelines. Follow the guidance described in Learn how to add continuous security validation to your CI/CD pipeline.

Next steps

View logs and alerts

Security logs and alerts using Azure services

Article • 11/30/2022

Logs provide insight into the operations of a workload, the infrastructure, network communications, and so on. When suspicious activity is detected, use alerts as a way of detecting potential threats. As part of your defense-in-depth strategy and continuous monitoring, respond to the alerts to prevent security assurance from decaying over time.

Key points

- ✓ Configure central security log management.
- ✓ Enable audit logging for Azure resources.
- ✓ Collect security logs from operating systems.
- ✓ Configure security log storage retention.
- ✓ Enable alerts for anomalous activities.

Use native services

- **Azure Monitor** provides observability across your entire environment. You automatically get platform metrics, activity logs, and diagnostics logs from most of your Azure resources with no configuration. The activity logs provide detailed diagnostic and auditing information.

- **Microsoft Defender for Cloud** generates notifications as security alerts by collecting, analyzings, and integrating log data from your Azure resources and the network. Alerts are available when you enable the Microsoft Defender plans. This will add to the overall cost.

- **Microsoft Sentinel** is a security information event management (SIEM) and security orchestration automated response (SOAR) solution. It's a single solution for alert detection, threat visibility, proactive hunting, and threat response.

Ideally use a combination of the preceding services to get a full view. For example, use Azure Monitor to collect information about the operating system running on Azure compute. If you're running your own compute, use Microsoft Defender for Cloud.

Audit logging

An important aspect of monitoring is tracking operations. For example, you want to know who created, updated, deleted a resource. Or, get resource-specific information such as when an image was pulled from Azure Container Registry. That information is crucial for a Security Operations (SecOps) team in detecting the presence of adversaries, reacting to an alert of suspicious activity, or proactively hunting for anomalous events. They are also useful for security auditing and compliance and offline analysis.

On Azure, that information is emitted as platform logs by the resources and the platform on which they run. They are tracked by Azure Resource Manager as and when subscription-level events occur. Each resource emits logs specific to the service.

Consider storing your data for audit purposes or statistical analysis. You can retain data in your log analytics workspace and specify the data type. This example sets the retention for `SecurityEvents` to 730 days:

HTTP

```http
PUT /subscriptions/00000000-0000-0000-0000-00000000000/resourceGroups/MyResourceGroupName/providers/Microsoft.OperationalInsights/workspaces/MyWorkspaceName/Tables/SecurityEvent?api-version=2017-04-26-preview {"properties":  {"retentionInDays": 730 } }
```

Retaining data in this manner can reduce your costs for data retention over time. For information about the type of data you can retain, see security data types.

Another way is to send the logs to a storage account.

Alerts

Security alerts are notifications that are generated when anomalous activity is detected on the resources used by the workload or the platform.

With the Microsoft Defender plans, Microsoft Defender for Cloud analyzes log data and shows a list of alerts that's based on logs collected from resources within a scope. Alerts include context information such as severity, status, activity time. Defender for Cloud also provides a correlated view called **incidents**. Use this data to analyze what actions the attacker took, and what resources were affected. Have strategies to react to alerts as soon as they are generated. An option is to handle alerts in Azure Functions.

Use the data to support these activities:

- Remediation of threats.
- Investigation of an incident.

- Proactive hunting activities.

For more information, see Security alerts and incidents.

Centralize logs and alerts

Organizations typically follow one of three models when deploying logs: centralized, decentralized, or hybrid. The choice depends on organizational structures. For example, if each team owns their resource group, log data is segregated per resource. While access control to that data might be easy to set up, it's difficult to correlate logs. This might be challenging for the SecOps team who need a holistic view to analyze the data.

Consider a central view of log and data, when applicable. Some advantages include:

- The resources in the workload can share a common log workspace reducing duplication.
- Single point of observability with all log data makes it easier consume data for hunting activities, querying, and statistical evaluation.
- The integrated data can be fed into modern machine learning analytics platforms support ingestion of large amounts of information and can analyze large datasets quickly. In addition, these solutions can be tuned to significantly reduce the false positive alerts.

You can collect logs and alerts from various sources centrally in a Log Analytics Workspace, storage account, and Event Hubs. You can then review and query log data efficiently. In Azure Monitor, use the **diagnostic setting** on resources to route specific logs that are important for the organization. Logs vary by resource type. In Microsoft Defender for Cloud, take advantage of the continuous export feature to route alerts.

> ⓘ **Note**
>
> Platform logs are not available indefinitely. You'll need to keep them so that you can review them later for auditing purposes or offline analysis. Use Azure Storage Accounts for long-term/archival storage. In Azure Monitor, specify a retention period when you enable diagnostic setting for your resources.

Another way to see all data in a single view is to integrate logs and alerts into Security Information and Event Management (SIEM) solutions, such as Microsoft Sentinel. Other popular third-party choices are Splunk, QRadar, ArcSight. Microsoft Defender for Cloud and Azure Monitor supports all of those solutions.

Integrating more data can enrich alerts with additional context. However, collection is not detection. Make sure a high volume of low value data doesn't flow into those solutions.

If you don't have a reasonable expectation that the data will provide value, deprioritize integration of these events. For example, high volume of firewall denies events may create noise without actual actions.

That choice will help in rapid response and remediation by filtering out false positives, and elevate true positives, and so on. Also it will lower SIEM cost, false positives, and increase performance.

Other ways of log integration may involve a hybrid model that mixes centralized and decentralized (distributed among teams) approaches. For details, see Important considerations for an access control strategy.

Next

Responding to alerts is an essential way to prevent security assurance decay, and designing for defense-in depth and least privilege strategies.

Remediate security risks

Related links

For more information, see these articles:

- How to get started with Azure Monitor and third-party SIEM integration
- How to collect platform logs and metrics with Azure Monitor
- Export alerts
- Understand Microsoft Defender for Cloud data collection

> Go back to the main article: Monitor

Remediate security risks in Microsoft Defender for Cloud

Article • 11/30/2022

Security controls must remain effective against attackers who continuously improve their ways to attack the digital assets of an enterprise. Use the principle of drive continuous improvement to make sure systems are regularly evaluated and improved.

Start by remediating common security risks. These risks are usually from well-established attack vectors. This will forces attackers to acquire use advanced and more expensive attack methods.

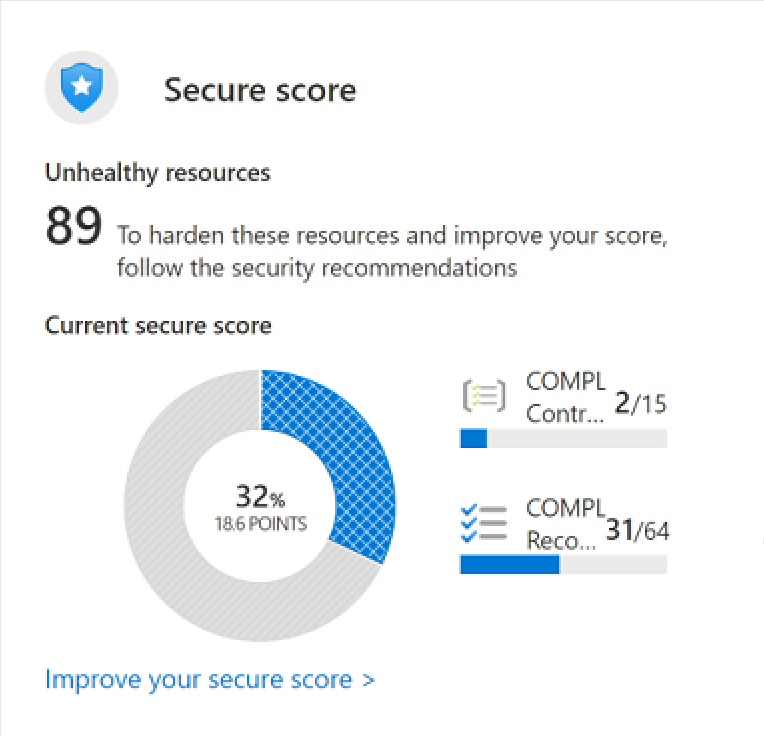
Key points

- Processes for handling incidents and post-incident activities, such as lessons learned and evidence retention.
- Remediate the common risks identified by Microsoft Defender for Cloud.
- Track remediation progress with secure score and comparing against historical results.
- Address alerts and take action with remediation steps.

Track Secure Score

Do you review and remediate common risks in the workload boundary?

Monitor the security posture of VMs, networks, storage, data services, and various other contributing factors. Secure Score in Microsoft Defender for Cloud shows a composite score that represents the security posture at the subscription level.



Do you have a process for formally reviewing Secure Score on Microsoft Defender for Cloud?

As you review the results and apply recommendations, track the progress and prioritize ongoing investments. Higher score indicates a better security posture.

- Set up a regular cadence (typically monthly) to review the secure score and plan initiatives with specific improvement goals.
- Assign stakeholders for monitoring and improving the score. Gamify the activity if possible to increase engagement and focus from the responsible teams.

As a technical workload owner, work with your organization's dedicated team that monitors Secure Score. In the DevOps model, workload teams may be responsible for their own resources. Typically, these teams are responsible.

- Security posture management team
- Vulnerability management or governance, risk, and compliance team
- Architecture team
- Resource-specific technical teams responsible for improving secure score, as shown in this table.

Category	Resources	Responsible team
Compute and applications	App Services	Application Development/Security Team(s)
	Containers	Application Development and/or Infrastructure/IT Operations

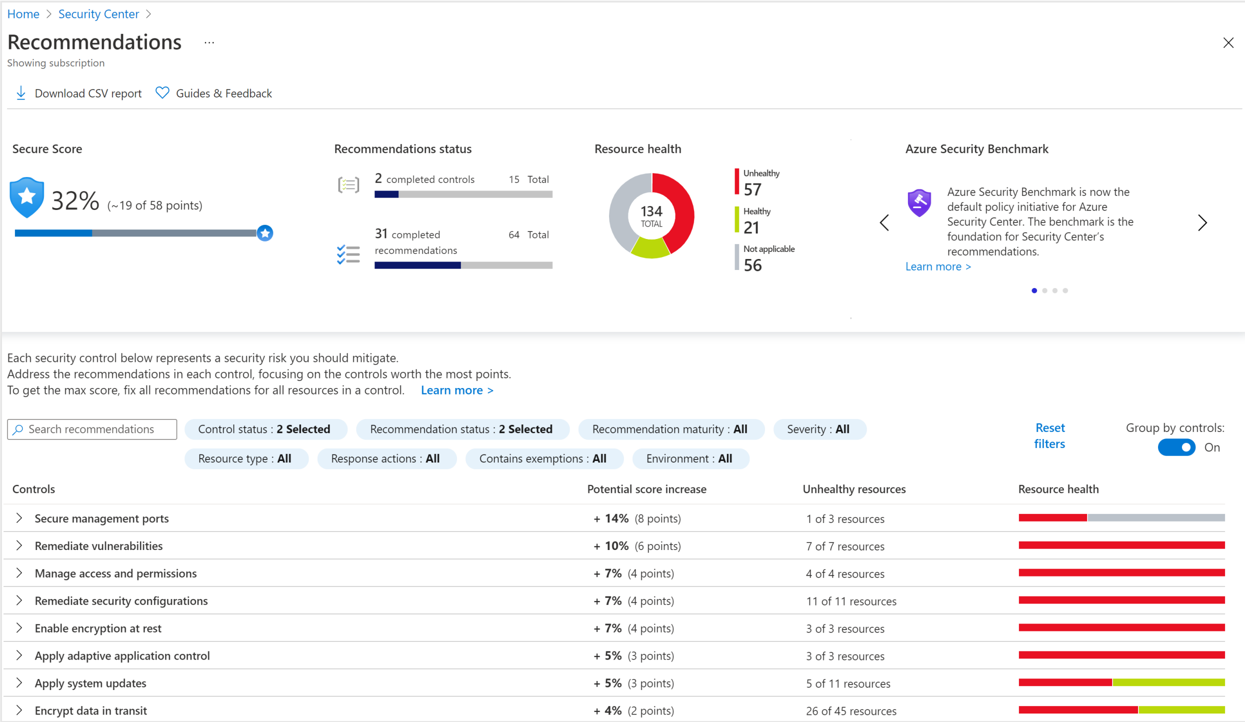
Category	Resources	Responsible team
	Virtual machines, scale sets, compute	IT/Infrastructure Operations
Data and Storage	SQL/Redis/Data Lake Analytics/Data Lake Store	Database Team
	Storage Accounts	Storage/Infrastructure Team
Identity and access management	Subscriptions	Identity Team(s)
	Key Vault	Information/Data Security Team
Networking Resources		Networking Team and Network Security Team
IoT Security	IoT Resources	IoT Operations Team

The Azure Secure Score sample shows how to get your Azure Secure Score for a subscription by calling the Microsoft Defender for Cloud REST API. The API methods provide the flexibility to query the data and build your own reporting mechanism of your secure scores over time.

Review and remediate recommendations

Microsoft Defender for Cloud monitors the security status of machines, networks, storage and data services, and applications to discover potential security issues. Enable this capability at no additional cost to detect vulnerable virtual machines connected to internet, missing security updates, missing endpoint protection or encryption, deviations from baseline security configurations, missing Web Application Firewall (WAF), and more.

View the recommendations to see the potential security issues and apply the Microsoft Defender for Cloud recommendations to execute technical remediations.



The recommendations are grouped by controls. Each recommendation has detailed information such as severity, affected resources, and quick fixes where applicable. Start with high severity items.

Defender for Cloud has the capability of exporting results at configured intervals. Compare the results with previous sets to verify that issues have been remediated.

For more information, see Continuous export.

Policy remediation

A common approach for maintaining the security posture is through Azure Policy.

Along with organizational policies, a workload owner can use scoped policies for governance purposes, such as check misconfiguration, prohibit certain resource types, and others. The resources are evaluated against rules to identify unhealthy resources that are risky. Post evaluation, certain actions are required as remediation. The actions can be enforced through Azure Policy effects.

For example, a workload runs in an Azure Kubernetes Service (AKS) cluster. The business goals require the workload to run in a highly restrictive environment. As a workload owner, you want the resource group to contain AKS clusters that are private. You can enforce that requirement with the **Deny** effect. It will prevent a cluster from being created if that rule isn't satisfied.

That sort of isolation can be maintained through policies at a higher level such as the subscription level or even management groups.

Another use case is that it can be automatically remediated by deploying related resources. For example, the organization wants all storage resources in a subscription to send logs to a common Log Analytics workspace. If a storage account doesn't pass the policy, a deployment is automatically started as remediation. That remediation can be enforced through **DeployIfNotExist**. There are some considerations.

- There's a significant wait before the resource is updated and the deployment starts. In the preceding example, there won't be logs captured during that wait time. Avoid using this effect for resources that cannot tolerate a delay.
- The resource deployed because of **DeployIfNotExist** are created by a separate identity than that of the identity that did the original deployment. That identity must have high enough privileges to make the required changes.

Manage alerts

Microsoft Defender for Cloud shows a list of alerts that's based on logs collected from resources within a scope. Alerts include context information such as severity, status, activity time. Most alerts have MITRE ATT&CK® tactics that can help you understand the kill chain intent. Select the alert and investigate the problem with detailed information.

Finally take action. That action can be to fix the resources that are out of compliance with actionable remediation steps. You can also suppress alerts that are false positives.

Make sure that you are integrating critical security alerts into Security Information and Event Management (SIEM), Security Orchestration Automated Response (SOAR) without introducing a high volume of low value data. Microsoft Defender for Cloud can stream alerts to Microsoft Sentinel. You can also use a third-party solution by using Microsoft Graph Security API.

Next

Azure security operations

Related links

> Go back to the main article: Monitor

Security audits

Article • 03/24/2023

To make sure that the security posture doesn't degrade over time, have regular auditing that checks compliance with organizational standards. Enable, acquire, and store audit logs for Azure services.

Key points

- ✓ Improve secure score in Microsoft Defender for Cloud.
- ✓ Use an industry standard benchmark to evaluate your organizations current security posture.
- ✓ Perform regular internal and external compliance audits, including regulatory compliance attestations.
- ✓ Review the policy requirements.
- ✓ Use Azure Governance Visualizer for a holistic overview of your technical Azure Governance implementation.

Evaluate using standard benchmarks

Do you evaluate the security posture of this workload using standard benchmarks?

Use an industry standard benchmark to evaluate your organizations current security posture.

Benchmarking allows you to improve your security program by learning from external organizations. It lets you know how your current security state compares to that of other organizations, providing both external validation for successful elements of your current system and identifying gaps that serve as opportunities to enrich your team's overall security strategy. Even if your security program isn't tied to a specific benchmark or regulatory standard, you will benefit from understanding the documented ideal states by those outside and inside of your industry.

As an example, the Center for Internet Security (CIS) has created security benchmarks for Azure that map to the CIS Control Framework. Another reference example is the MITRE ATT&CK™ framework that defines the various adversary tactics and techniques based on real-world observations. These external references control mappings and help you to understand any gaps between your current strategy, what you have, and what other experts have in the industry.

Suggested action

Develop an Azure security benchmarking strategy aligned to industry standards.

As people in the organization and on the project change, it is crucial to make sure that only the right people have access to the application infrastructure. Auditing and reviewing access control reduces the attack vector to the application. Azure control plane depends on Azure AD and access reviews are often centrally performed as part of internal, or external audit activities.

Make sure that the security team is auditing the environment to report on compliance with the security policy of the organization. Security teams may also enforce compliance with these policies.

Audit regulatory compliance

Compliance is important for several reasons. Aside from signifying levels of standards, like `ISO 27001` and others, noncompliance with regulatory guidelines may bring sanctions and penalties. Regularly review roles that have high privileges. Set up a recurring review pattern to ensure that accounts are removed from permissions as roles change. Consider auditing at least twice a year.

Suggested action

Use Microsoft Defender for Cloud to continuously assess and monitor your compliance score.

Learn more

[Assess your regulatory compliance]

Have you established a monitoring and assessment solution for compliance?

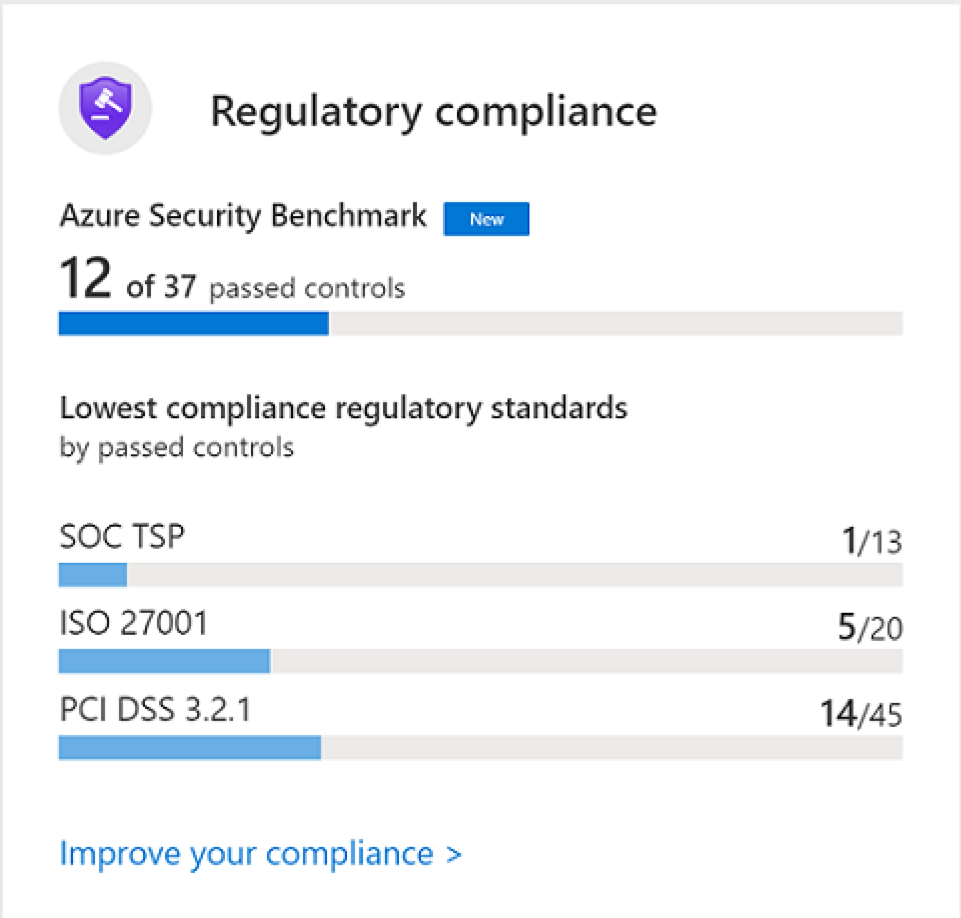
Continuously assess and monitor the compliance status of your workload. Microsoft Defender for Cloud provides a regulatory compliance dashboard that shows the current security state of workload against controls mandated by the standard governments or industry organizations and Azure Security Benchmark. Keep your resources in compliance with those standards. Defender for Cloud tracks many standards. You can set the standards by management groups in a subscription.

Consider using Azure Access Reviews or Entitlement Management to periodically review access to the workload.

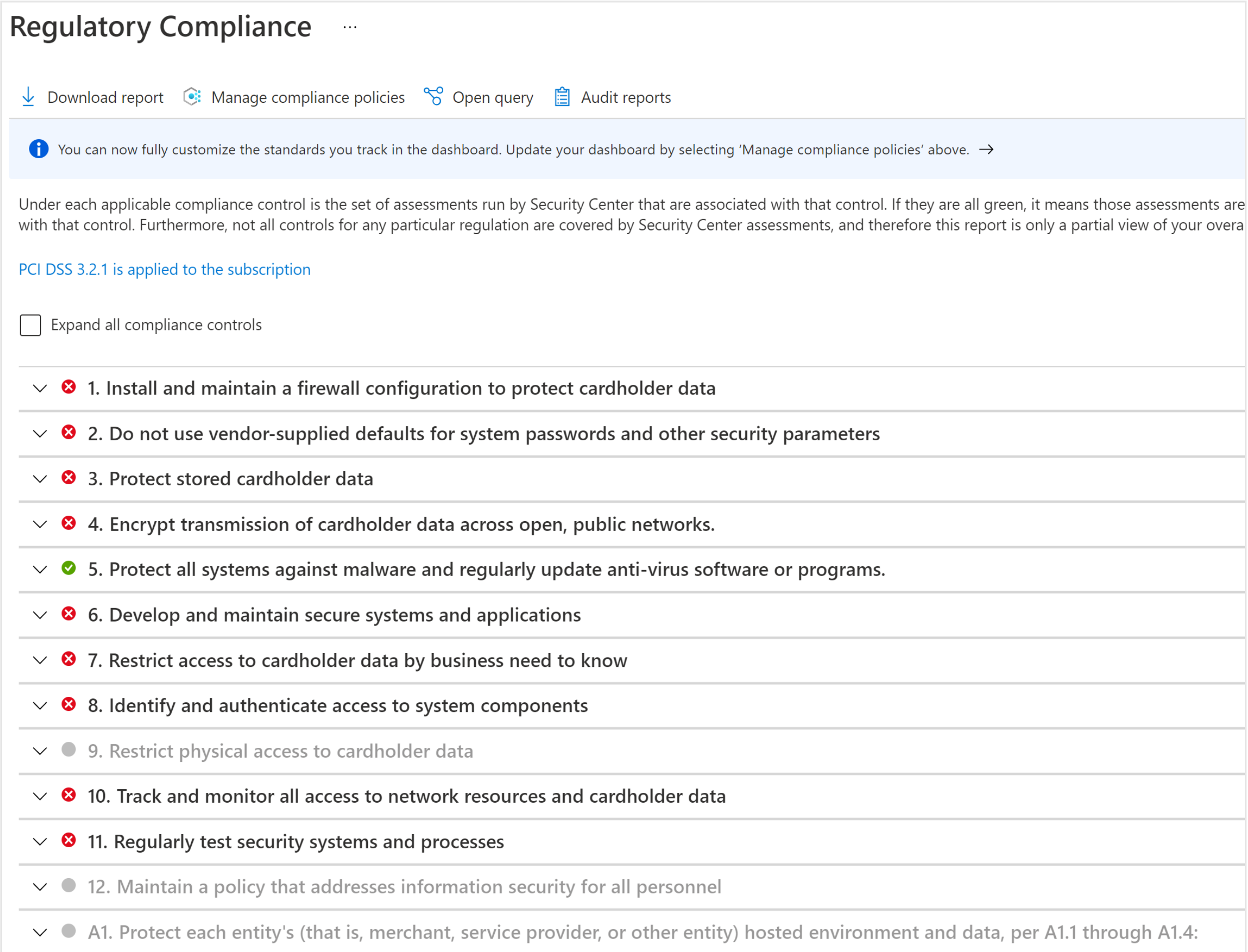
Consider using Azure Access Reviews or Entitlement Management to periodically review access to the workload.

For Azure, use Azure Policy to create and manage policies that enforce compliance. Azure Policies are built on the Azure Resource Manager capabilities. Azure Policy can also be assigned through Azure Blueprints.

For more information, see Tutorial: Create and manage policies to enforce compliance.



Here's an example management group that is tracking compliance to the Payment Card Industry (PCI) standard.



Do you have internal and external audits for this workload?

A workload should be audited internally, external, or both with the goal of discovering security gaps. Make sure that the gaps are addressed through updates.

Auditing is important for workloads that follow a standard. Aside from signifying levels of standards, noncompliance with regulatory guidelines may bring sanctions and penalties.

Perform regulatory compliance attestation. Attestations are done by an independent party that examines if the workload is in compliance with a standard.

Review critical access

Is access to the control plane and data plane of the application periodically reviewed?

Regularly review roles that have high privileges. Set up a recurring review pattern to ensure that accounts are removed from permissions as roles change. Consider auditing at least twice a year.

As people in the organization and on the project change, make sure that only the right people, have access to the application infrastructure and just enough privileges to complete the task. Auditing and reviewing the access control reduces the attack vector to the application.

Azure control plane depends on Azure AD. You can conduct the review manually or through an automated process by using tools such as Azure AD access reviews. These reviews are often centrally performed often as part of internal or external audit activities.

Check policy compliance

Make sure that the security team is auditing the environment to report on compliance with the security policy of the organization. Security teams may also enforce compliance with these policies.

Enforce and audit industry, government, and internal corporate security policies. Policy monitoring checks that initial configurations are correct and that it continues to be compliant over time.

For Azure, use Azure Policy to create and manage policies that enforce compliance. Azure Policies are built on the Azure Resource Manager capabilities. Azure Policy can also be assigned through Azure Blueprints. For more information, see Tutorial: Create and manage policies to enforce compliance.

Capture critical data

Azure Governance Visualizer captures data from the most relevant Azure governance capabilities such as Azure Policy, Azure role-based access control (Azure RBAC), and Azure Blueprints. The visualizer PowerShell script iterates through an Azure tenant's management group hierarchy down to the subscription level. From the collected data, the visualizer shows your hierarchy map, creates a tenant summary, and builds granular scope insights about your management groups and subscriptions.

Next steps

Remediate security risks in Microsoft Defender for Cloud

Related links

- Secure score in Microsoft Defender for Cloud allows you view all the security vulnerabilities into a single score.

- Tutorial: Improve your regulatory compliance describes a step-by-step process to evaluate regulatory requirements in Microsoft Defender for Cloud.

Azure security test practices

Article • 03/24/2023

Regularly test your security design and implementation, as part the organization's operations. That integration will make sure the security assurances are effective and maintained as per the security standards set by the organization.

A well-architected workload should be resilient to attacks. It should recover rapidly from disruption and yet provide the security assurances of confidentiality, integrity, and availability. Invest in simulated attacks as tests that can indicate gaps. Based on the results of the results you can harden the defense and limit a real attacker's lateral movement within your environment.

Simulated tests can also give you data to plan risk mitigation. Applications that are already in production should use data from real-world attacks. New or updated applications with new features, should rely on structured models for detecting risks early, such as threat modeling.

Key points

- Define test cases that are realistic and based on real-world attacks.
- Identify and catalog lowest cost methods for preventing and detecting attacks.
- Use penetration testing as a one-time attack to validate security defenses.
- Simulate attacks through red teams for long-term persistent attacks.
- Measure and reduce the potential attack surface that attackers target for exploitation for resources within the environment.
- Ensure proper follow-up to educate users about the various means that an attacker may use.

Penetration testing (pentesting)

Do you perform penetration testing on the workload?

It's recommended that you simulate a one-time attack to detect vulnerabilities. Pentesting is a popular methodology to validate the security defense of a system. The practitioners are security experts who are not part of the organization's IT or application teams. So, they look at the system in a way that malicious actors scope an attack surface. The goal is to find security gaps by gathering information, analyzing vulnerabilities, and reporting.

Penetration tests provide a point-in-time validation of security defenses. Red teams can help provide ongoing visibility and assurance that your defenses work as designed, potentially testing across different levels within your workload(s). Red team programs can be used to simulate either one time, or persistent threats against an organization to validate defenses that have been put in place to protect organizational resources.

Microsoft recommends penetration testing and red team exercises to validate security defenses for your workload.

Penetration Testing Execution Standard (PTES) provides guidelines about common scenarios and the activities required to establish a baseline.

Azure uses shared infrastructure to host your assets and assets belonging to other customers. In a pentesting exercise, the practitioners may need access to sensitive data of the entire organization. Follow the rules of engagement to make sure that access and the intent is not misused. For guidance about planning and executing simulated attacks, see Penetration Testing Rules of Engagement .

Learn more

- Azure Penetration Testing
- Penetration Testing

Simulate attacks

The way users interact with a system is critical in planning your defense. The risks are even higher for critical impact accounts because they have elevated permissions and can cause more damage.

Do you carry out simulated attacks on users of this workload?

Simulate a persistent threat actor targeting your environment through a red team. Here are some advantages:

- Periodic checks. The workload will get checked through a realistic attack to make sure the defense is up to date and effective.
- Educational purposes. Based on the learnings, upgrade the knowledge and skill level. This will help the users understand the various means that an attacker may use to compromise accounts.

A popular choice to simulate realistic attack scenarios is Office 365 Attack Simulator.

Is personal information detected and removed/obfuscated automatically?

Be cautious about using sensitive application information. Don't store personal information such as contact information, payment information, and so on, in any application logs. Apply protective measures, such as obfuscation. Machine learning tools can help with this measure. For more information, see PII Detection cognitive skill.

Related links

Threat modeling is a structured process to identify the possible attack vectors. Based on the results, prioritize the risk mitigate efforts. For more information, see Application threat analysis.

For more information on current attacks, see the Microsoft Security Intelligence (SIR) report.

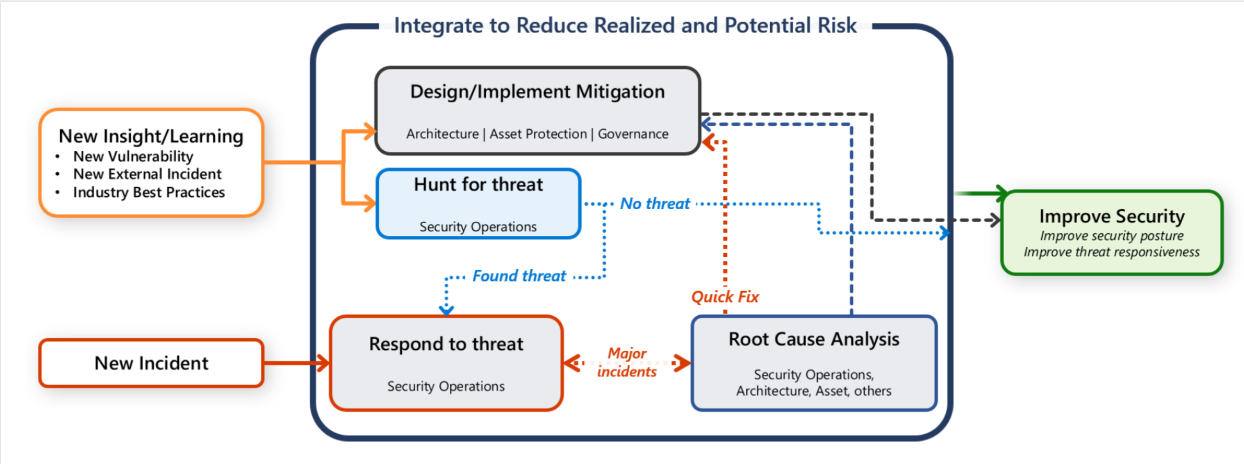
Microsoft Cloud Red Teaming

> Go back to the main article: Monitor

Security operations in Azure

Article • 04/10/2023

The responsibility of the security operation team (also known as Security Operations Center (SOC), or SecOps) is to rapidly detect, prioritize, and triage potential attacks. These operations help eliminate false positives and focus on real attacks, reducing the mean time to remediate real incidents. Central SecOps team monitors security-related telemetry data and investigates security breaches. It's important that any communication, investigation, and hunting activities are aligned with the application team.



Here are some general best practices for conducting security operations:

- Follow the NIST Cybersecurity Framework functions as part of operations.
  - **Detect** the presence of adversaries in the system.
  - **Respond** by quickly investigating whether it's an actual attack or a false alarm.
  - **Recover** and restore the confidentiality, integrity, and availability of the workload during and after an attack.

  For information about the framework, see NIST Cybersecurity Framework .

- Acknowledge an alert quickly. A detected adversary must not be ignored while defenders are triaging false positives.
- Reduce the time to remediate a detected adversary. Reduce their opportunity time to conduct and attack and reach sensitive systems.
- Prioritize security investments into systems that have high intrinsic value. For example, administrator accounts.

- Proactively hunt for adversaries as your system matures. This effort will reduce the time that a higher skilled adversary can operate in the environment. For example, skilled enough to evade reactive alerts.

For information about the metrics that the Microsoft's SOC team uses , see Microsoft SOC .

Tools

Here are some Azure tools that a SOC team can use investigate and remediate incidents.

Tool	Purpose
Microsoft Sentinel	Centralized Security Information and Event Management (SIEM) to get enterprise-wide visibility into logs.
Microsoft Defender for Cloud	Alert generation. Use security playbook in response to an alert.
Azure Monitor	Event logs from application and Azure services.
Azure Network Security Group (NSG)	Visibility into network activities.
Azure Information Protection	Secure email, documents, and sensitive data that you share outside your company.

Investigation practices should use native tools with deep knowledge of the asset type such as an Endpoint detection and response (EDR) solution, Identity tools, and Microsoft Sentinel.

For more information about monitoring tools, see Security monitoring tools in Azure.

Assign incident notification contact

Security alerts need to reach the right people in your organization. Establish a designated point of contact to receive Azure incident notifications from Microsoft Defender for Cloud. In most cases, such notifications indicate that your resource is compromised or attacking another customer. This enables your security operations team to rapidly respond to potential security risks and remediate them.

This enables your security operations team to rapidly respond to potential security risks and remediate them.

Ensure administrator contact information in the Azure enrollment portal includes contact information that will notify security operations directly or rapidly through an internal process.

Learn more

To learn more about establishing a designated point of contact to receive Azure incident notifications from Microsoft, reference the following articles:

- Update notification settings
- Configure email notifications for security alerts

Incident response

Is the organization effectively monitoring security posture across workloads, with a central SecOps team monitoring security-related telemetry data and investigating possible security breaches? Communication, investigation, and hunting activities need to be aligned with the application team(s).

Are operational processes for incident response defined and tested?

Actions executed during an incident and response investigation could impact application availability or performance. Define these processes and align them with the responsible (and in most cases central) SecOps team. The impact of such an investigation on the application has to be analyzed.

Are there tools to help incident responders quickly understand the application and components to do an investigation?

Incident responders are part of a central SecOps team and need to understand security insights of an application. Security playbook in Microsoft Sentinel can help to understand the security concepts and cover the typical investigation activities.

Suggested action

Consider using Microsoft Defender for Cloud to monitor security-related events and get alerted automatically.

Learn more

Security alerts and incidents in Microsoft Defender for Cloud

Hybrid enterprise view

Security operations tooling and processes should be designed for attacks on cloud and on-premises assets. Attackers don't restrict their actions to a particular environment when targeting an organization. They attack resources on any platform using any method available. They can pivot between cloud and on-premises resources using identity or other means. This enterprise-wide view will enable SecOps to rapidly detect, respond, and recover from attacks, reducing organizational risk.

Leverage native detections and controls

Use Azure security detections and controls instead of creating custom features for viewing and analyzing event logs. Azure services are updated with new features and have the ability to detect false positive with a higher accuracy rate.

Integrating logs from the network devices, and even raw network traffic itself, will provide greater visibility into potential security threats flowing over the wire.

To get a unified view across the enterprise, feed the logs collected through native detections (such as Azure Monitor) into a centralized security information and event management (SIEM) solution like Microsoft Sentinel. Avoid using generalized log analysis tools and queries. Within Azure Monitor, create Log Analytics Workspace to store logs. You can also review logs and perform queries on log data. These tools can offer high-quality alerts.

The modern machine learning-based analytics platforms support ingestion of extremely large amounts of information and can analyze large datasets very quickly. In addition, these solutions can be tuned to significantly reduce false positive alerts.

Examples of network logs that provide visibility include:

- Security group logs - flow logs and diagnostic logs
- Web application firewall logs
- Virtual network taps and their equivalents
- Azure Network Watcher

Suggested actions

Integrate network device log information in advanced SIEM solutions or other analytics platforms.

Learn more

Enable enhanced network visibility

Next steps

- Security health modeling
- Security tools
- Security logs and audits
- Check for identity, network, data risks

Tradeoffs for security

Article • 11/30/2022

Security provides confidentiality, integrity, and availability assurances of an organization's data and systems. When designing a system you can almost never compromise on security controls. When you enhance security of an architecture there might be impact on reliability, performance efficiency, cost, and operational excellence. This article describes some of those considerations.

Security vs Reliability

Reliable applications are resilient and highly available. Every architectural component factors in achieving your requirements for reliability. Workload security is often woven into many layers of the workload's architecture, operations, and runtime requirements; and may come with their own implications on resiliency or availability.

For example, identity providers and authorization services are critical dependencies to consider. This includes the identity service (Microsoft Identity Platform) and any libraries that help facilitate the use of those services. At some points in the architecture, a failure at an identity layer is terminal. At other points, reliability can be still achieved through strategies such as caching, taking advantage of TTLs on access tokens, and others. OAuth2 claims validation can happen mostly disconnected from the claims provider. However, not all authorization can be achieved that way. In those situations reliability may be traded in favor of complete security.

Many workloads may quickly degrade in functionality with the loss of critical security controls. Consider evaluating at each component of your architecture to detect that condition.

Other security considerations that might impact reliability are:

- Poor or manual certifications or key rotation practices. Failure to do those tasks can lead to reliability issues.
- Expired service principals. For example, a deployment pipeline that used a service principal might fail at a later date, if that principal's access key has expired. Using managed identities helps keep reliability high while also maintaining least privileges on that identity.
- High availability is often achieved by redundancy (actively or passively), and security controls also need to align with the failover mechanism. For example, failing over from one storage account to another for reliability may impact how the client's active authorization session is handled. Using managed identity with Azure

AD integration for storage access can result in a higher reliability because the client doesn't have to manage SAS tokens when switching to the new storage account.

Security vs Cost Optimization

Increasing security of the workload will almost always lead to higher cost. There are some ways to optimize cost.

- Maximum security may not always be practical for all environments. Evaluate the security requirements in pre-production and production environments. Are services such as Azure DDoS Protection, Microsoft Sentinel, Dedicated HSMs, Microsoft Defender for Cloud needed in pre-production? Is inner loop mocking of security controls sufficient? If resources are not publicly accessible, can you dial down some controls for cost savings? Always make those choices, *if and only if*, the lowered environment still meets the business requirements.

- Premium security features can increase the cost. There are areas you can reduce cost by using native security features. For example, avoid implementing custom roles if you can use built-in roles.

- Every security control has an opportunity to impact workflows, and workflows that involve people can be expensive. A security control that stops work from being done should be evaluated as necessary or unnecessarily redundant. Total cost of ownership (TCO) includes operational costs for developers, operators, IT SecOps and onerous security protocols. Reach agreement about where "less" can br "sufficient" to optimize costs.

- TCO includes the time needed do tasks. Optimizing that time will optimize cost. Using platform features can lower TCO and enhance the security posture. Instead of training an engineer to manually review logs and correlate access patterns, use intelligence in services, such as Microsoft Defender for Cloud or Sentinel alerts.

Security vs Operational Excellence

Operational Excellence involves understanding business and workload behavior, and applying appropriate amounts of automation, and observability into those processes.

- Release management

  Your organization defines its quality gates (manual or automated) as part of its safe deployment practices. Evaluate whether each gate is required or optional, taking into consideration whether it's valuable to perform, at the added cost of complexity (and time) for that release. Adding security checks into the process makes the checks more valuable at that point in the process than any gains (usually simplicity and time) by not having them in place.

- Organizational policies

  Teams can often benefit from collaborating and using cross-functional skills in all stages of workload life cycle, from development all the way through production. However, organizational policy or regulatory concerns may prevent such wide-reaching access. Consider, where possible, isolating those systems that do need heightened access policies from those that do not. Avoid applying "one size fits all" model to all components in the system. It's easier to optimize operations in systems that allow more unregulated access. For systems that demand regulated access, complexity is expected to increase leading to higher cost.

- Supportability considerations

  The most "serviceable" architectures are the ones that are the most transparent to everyone involved, and those often have the least number of security controls. Adding security controls to your architecture like filtered telemetry feeds, redacted logs, runtime system access restrictions, and so on can all impact the supportability of a solution. Adding security controls often require adding compensating or compromised solutions for observability into the platform.

Related link

> Go back to the main article: Security

Cost optimization documentation

Apply the cost principles in your architecture to accelerate your time to market while avoiding capital-intensive solutions. Consider opportunity costs. Use the cost calculators to estimate the initial and operational costs. Finally, establish policies, budgets, and controls that set cost limits for your solution.

Key points

QUICKSTART

Principles

Design checklist

Provision checklist

Monitor checklist

Optimize checklist

Tradeoffs

TRAINING

Cost optimization

VIDEO

What trade-offs have you made to optimize for cost?

Design within the cost limits

GET STARTED

Organization structure

Capture clear requirements

Consider the cost constraints

Consider tradeoffs

CONCEPT

Develop a cost model

Develop a cost model

Understand the usage meters

Cost impact across regions

Consumption or fixed?

Use PaaS

Estimate an initial cost

VIDEO

How is your organization modeling cloud costs?

Provision the right resources

OVERVIEW

AI + Machine Learning

Big Data

Data store

Networking

VIDEO

How do you ensure that cloud resources are appropriately provisioned?

How do you manage the storage footprint of your digital assets?

Monitor your cloud spend

CONCEPT

Review alerts

Generate cost reports

Conduct cost reviews

VIDEO

How are you monitoring your costs?

Optimize based on audits

CONCEPT

Autoscale instances

Right-size VMs

Cache data

VIDEO

What actions are you taking to optimize cloud costs?

Cost tools and services

CONCEPT

Azure Pricing Calculator

Azure Migrate

Microsoft Azure Total Cost of Ownership (TCO) Calculator

Azure Advisor

Advisor Score

Azure Cost Management

Visualize cost reports

Consumption APIs

REFERENCE

Billing account API

Billing Periods API

Usage Detail API

Marketplace Store Charge API

Price Sheet API

Overview of the cost optimization pillar

Article • 03/30/2023

The cost optimization pillar provides principles for balancing business goals with budget justification. The principles help you create a cost-effective workload while avoiding capital-intensive solutions. Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies.

Use the pay-as-you-go strategy for your architecture, and invest in scaling out, rather than delivering a large investment-first version. Consider opportunity costs in your architecture, and the balance between *first mover* advantage versus *fast follow*. Use the cost calculators to estimate the initial cost and operation costs. Finally, establish policies, budgets, and controls that set cost limits for your solution.

To assess your workload using the tenets found in the Microsoft Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.

We recommend exploring the following videos to dive deeper into Azure cost optimization:

https://learn.microsoft.com/shows/Azure-Enablement/Diving-deeper-into-Azure-cost-optimization-Part-1-Cost-Optimization-Ep-2-Well-Architected-series/player

https://learn.microsoft.com/shows/Azure-Enablement/Diving-deeper-into-Azure-cost-optimization-Part-2-Cost-Optimization-Ep-2-Well-Architected-series/player

Azure Well-Architected Framework cost optimization articles

The Azure Well-Architected Framework includes the following articles in the cost optimization pillar:

Cost area	Description
Capture cost requirements	Start your planning with a careful enumeration of requirements. Make sure the needs of the stakeholders are addressed. For strong alignment with business goals, the stakeholders must define the needs not the vendors.
Cost of resources in Azure regions	Cost of an Azure service can vary between locations based on demand and local infrastructure costs.

Cost area	Description
Governance	Understand how governance can assist with cost management. This work benefits your ongoing cost review process and offers a level of protection for new resources.
Estimate the initial cost	It's difficult to attribute costs before deploying a workload to the cloud. If you use methods for on-premises estimation or directly map on-premises assets to cloud resources, the estimate is inaccurate.
PaaS	Look for areas in the architecture where it might be natural to incorporate platform-as-a-service (PaaS) options. These options include caching, queues, and data storage. PaaS reduces time and cost of managing servers, storage, networking, and other application infrastructure.
Consumption	A common way to estimate cost is by considering workloads on a peak throughput. Under consistently high usage, consumption-based pricing can be less efficient for estimating baseline costs when compared to the equivalent provisioned pricing.
Provision cloud resources	Deployment of workload cloud resources can optimize cost.
Monitor cost	Azure Cost Management has an alert feature. Alerts are generated when consumption reaches a threshold.
Optimize cost	Monitor and optimize the workload by using the right resources and sizes.
Tradeoffs for costs	As you design the workload, consider tradeoffs between cost optimization and other aspects of the design, such as security, scalability, resilience, and operability.

Next section

Use the cost optimization principles to guide you in your overall strategy.

Principles

Cost optimization design principles

Article • 04/18/2023

Business goals and the return on investment (ROI) drive a cost-effective workload while you keep to a given budget. The principles of cost optimization are important considerations that help you achieve both business objectives and cost justification.

To assess your workload using the tenets you find in the Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.

The following design principles provide:

- Context for questions
- Why a certain aspect is important
- How an aspect applies to cost optimization

Use these critical design principles as lenses to assess the cost optimization of an application deployed on Azure. These lenses provide a framework for the application assessment questions.

Choose the correct resources

Choose resources that align with business goals and can handle your workload performance.

When onboarding new workloads, explore the possibility of modernization and cloud native offerings where possible. It's typically more cost-effective to use platform as a service (PaaS) or software as a service (SaaS), as opposed to infrastructure as a service (IaaS).

An inappropriate or misconfigured service can affect cost. For example, building a multi-region service when the service levels don't require high-availability or geo-redundancy increases cost without any reasonable business justification.

Set up budgets and maintain cost constraints

Every design choice has cost implications. Consider the budget constraints set by the company before choosing:

- An architectural pattern
- The Azure service

- A price model for the service

As part of design, identify acceptable boundaries on:

- Scale
- Redundancy
- Performance against cost

After estimating the initial cost, set budgets and alerts at different scopes to measure the cost. One cost driver can be unrestricted resources. These resources typically need to scale and consume more cost to meet demand.

Dynamically allocate and deallocate resources

To match performance needs, dynamically allocate and deallocate resources.

Identify idle or underutilized resources through Azure Advisor or other tools, and:

- Reconfigure
- Consolidate (or)
- Shut down

Optimize workloads, aim for scalable costs

A key benefit of the cloud is the ability to scale dynamically. The workload cost should scale linearly with demand. You can save cost through automatic scaling.

Recommendations:

- Consider usage metrics and performance to determine the number of instances.
- Choose smaller instances for a highly variable workload.
- Scale out, rather than up, to get the required level of performance. This choice enables you to make your cost calculations and estimates granular.

The cost management process should be:

- Rigorous
- Iterative
- A key principle of responsible cloud optimization

Continuously monitor and optimize cost management

To provision resources dynamically and to scale with demand:

- Conduct regular cost reviews
- Measure capacity needs
- Forecast capacity needs

If you're just starting this process, review enable success during a cloud adoption journey.

Next step

Design checklist

Checklist - Design for cost

Article • 11/30/2022

Use this checklist when designing a cost-effective workload.

Cost model

- **Capture clear requirements**. Gather detailed information about the business workflow, regulatory, security, and availability.
  - Capture requirements
- **Estimate the initial cost**. Use tools such as Azure pricing calculator to assess cost of the services you plan to use in the workload. Use Azure Migrate and Microsoft Azure Total Cost of Ownership (TCO) Calculator for migration projects. Accurately reflect the cost associated with right storage type. Add hidden costs, such as networking cost for large data download.
  - Estimate the initial cost
- **Define policies for the cost constraints defined by the organization**. Understand the constraints and define acceptable boundaries for quality pillars of scale, availability, security.
  - Consider the cost constraints
- **Identify shared assets**. Evaluate the business areas where you can use shared resources. Review the billing meters build chargeback reports per consumer to identify metered costs for shared cloud services.
  - Create a structured view of the organization in the cloud
- **Plan a governance strategy**. Plan for cost controls through Azure Policy. Use resource tags so that custom cost report can be created. Define budgets and alerts to send notifications when certain thresholds are reached.
  - Governance

Architecture

- **Check the cost of resources in various Azure geographic regions**. Check your egress and ingress cost, within regions and across regions. Only deploy to multiple regions if your service levels require it for either availability or geo-distribution.
  - Azure regions

- **Choose a subscription that is appropriate for the workload**. Azure Dev/Test subscription types are suitable for experimental or non-production workloads and have lower prices on some Azure services such as specific VM sizes. If you can commit to one or three years, consider subscriptions and offer types that support Azure Reservations or savings plans.
  - Subscription and offer type

- **Choose the right resources to handle the performance**. Understand the usage meters and the number of meters for each resource in the workload. Consider tradeoffs over time. For example, cheaper virtual machines may initially indicate a lower cost but can be more expensive over time to maintain a certain performance level. Be clear about the billing model of third-party services.
  - Azure resources
  - Use cost alerts to monitor usage and spending

- **Compare consumption-based pricing with pre-provisioned cost**. Establish baseline cost by considering the peaks and the frequency of peaks when analyzing performance.
  - Consumption and fixed cost models

- **Use proof-of-concept deployments**. The Azure Architecture Center has many reference architectures and implementations that can serve as a starting point. The Azure Tech Community has architecture and services forums.

- **Choose managed services when possible**. With PaaS and SaaS options, the cost of running and maintaining the infrastructure is included in the service price.
  - Managed services

Develop a cost model

Article • 04/10/2023

Cost modeling is an exercise where you create logical groups of cloud resources that are mapped to the organization's hierarchy and then estimate costs for those groups. The goal of cost modeling is to estimate the overall cost of the organization in the cloud.

1. **Understand how your responsibilities align with your organization**

   Map your organization's needs to logical groupings offered by cloud services. This way the business leaders of the company get a clear view of the cloud services and how they're controlled.

2. **Capture clear requirements**

   Start your planning by carefully enumerating the requirements. From the high-level requirements, narrow down each requirement before starting on the design of the solution.

3. **Consider the cost constraints**

   Evaluate the budget constraints on each business unit. Determine the governance policies in Azure to lower cost by reducing wastage, over-provisioning, or expensive provisioning of resources.

4. **Consider tradeoffs**

   Optimal design doesn't equate to a lowest-cost design.

   As you prioritize requirements, cost can be adjusted. Expect a series of tradeoffs in the areas that you want to optimize, such as security, scalability, resilience, and operability. If the cost to address the challenges in those areas is high, stakeholders look for alternate options to reduce cost. There might be risky choices made in favor of a cheaper solution.

5. **Derive functional requirements from high-level goals**

   Break down the high-level goals into functional requirements for the solution's components. Each requirement must be based on realistic metrics to estimate the actual cost of the workload.

6. **Consider the billing model for Azure resources**

Azure services are offered with consumption-based prices where you're charged for only what you use. There's also options for fixed price where you're charged for provisioned resources.

Most services are priced based on units of size, amount of data, or operations. Understand the meters that are available to track usage. For more information, see Azure resources.

At the end of this exercise, you should have identified the lower and upper limits on cost and set budgets for the workload. Azure lets you create and manage budgets in Azure Cost Management. For more information, see Quickstart: Create a budget with an Azure Resource Manager template.

Have frequent and clear communication with the stakeholders

In the initial stages, communication between stakeholders is vital. The overall team must align on the requirements so that overall business goals are met. If not, the entire solution might be at risk. For instance, the development team indicates that the resilience of a monthly batch-processing job is low. They might request the job to work as a single node without scaling capabilities. This request opposes the architect's recommendation to automatically scale out and route requests to worker nodes.

This type of disagreement can introduce a point of failure into the system, risking the service level agreement (SLA), and cause an increase in operational cost.

Organization structure

Map the organization's needs to logical groupings offered by cloud services. This way the business leaders of the company get a clear view of the cloud services and how they're controlled.

1. Understand how your workload fits into cost optimization across the portfolio of cloud workloads.

   If you're creating a workload that fits into a broader portfolio of workloads, see the get started guide to document foundational decisions. The guide helps your team capture the broader portfolio view of business units, resources organizations, responsibilities, and a view of the long-term portfolio.

   If a central team needs to run cost optimization across the portfolio, see get started managing enterprise costs in Cloud Adoption Framework.

2. Encourage a culture of democratized cost optimization decisions.

As a workload owner, you can have a measurable effect on cost optimization. There are other roles in the organization that can help improve cost management activities. To help embed the pillar of cost optimization into your organization beyond your workload team, see Build a cost-conscious organization.

3. Reduce costs through shared cloud services and landing zones.

   If your workload has dependencies on shared assets like Active Directory, Network connectivity, security devices, or other services that are also used by other workloads, encourage your central IT organization to provide those services through a centrally managed landing zone to reduce duplicate costs. See get started with centralized design and configuration to start developing landing zones.

4. Calculate the ROI by understanding what's included in each grouping and what isn't.

   Which aspects of the hierarchy are covered by cloud services?

   The Azure pricing model is based on expenses incurred for the service. Expenses include hardware, software, development, operations, security, and data center space to name a few. Evaluate the cost benefit of shifting away from owned technology infrastructure to leased technology solutions.

5. Identify scenarios where you can use shared cloud services to lower cost.

   Can some services be shared by other consumers?

   Identify areas where you can share a service or an application environment with other business units.

   Identify resources that you can use as shared services and review their billing meters. Examples include a virtual network and its hybrid connectivity or a shared app service environment (ASE). If the meter data isn't able to be split across consumers, decide on custom solutions to allocate proportional costs. Move shared services to dedicated resources for consumers for cost reporting.

   > Build chargeback reports per consumer to identify metered costs for shared cloud services. Aim for granular reports to understand which workload is consuming what amount of the shared cloud service.

Next step

Capture cost requirements

Cost constraints

Here are some considerations for determining the governance policies that can assist with cost management.

- What are the budget constraints set by the company for each business unit?
- What are policies for the budget alert levels and associated actions?
- Identify acceptable boundaries for scale, redundancy, and performance against cost.
- Assess the limits for security. Don't compromise on security. Premium cloud security features can drive the cost up. It's not necessary to overinvest. Instead, use the cost profile to drive a realistic threat profile.
- Identify unrestricted resources. These resources typically need to scale and consume more cost to meet demand.

Next step

Consider tradeoffs

Functional requirements

Break down high-level goals into functional requirements. For each of those requirements, define metrics to calculate cost estimates accurately. Cloud services are priced based on performance, features, and locations. When defining these metrics, identify acceptable boundaries of performance, scale, resilience, and security. Start by expressing your goals in number of business transactions over time, breaking them down to fine-grain requirements.

What resources are needed for a single transaction, and how many transactions are done per second, day, and year?

> Start with a fixed cost of operations and a rough estimate of transaction volume to work out a cost-per-transaction to establish a baseline. Consider the difference between cost models based on fixed, static provisioning of services, more variable costs based upon autoscaling such as serverless technologies.

Use T-shirt sizes for choosing SKUs

When choosing options for services, start with an abstract representation of size. For example, if you choose a T-shirt size approach, small, medium, and large sizes can represent an on-demand virtual machine. You pick a T-shirt size instead of picking specific virtual machines or managed disks SKU sizes.

Abstract sizes give you an idea of the expected performance and cost requirements. It sets the tone for various consumption units that measure compute resources for performance. Also, it helps in understanding the on-demand consumption model of the services.

For more information, see Estimate the initial cost.

Next steps

Estimate the initial cost

Capture cost requirements

Article • 11/30/2022

Start your planning with a careful enumeration of requirements. Make sure the needs of the stakeholders are addressed. For strong alignment with business goals, those areas must be defined by the stakeholders and shouldn't be collected from a vendor.

Capture requirements at these levels:

- Business workflow
- Compliance and regulatory
- Security
- Availability

What do you aim to achieve by building your architecture in the cloud?

Here are some common answers.

- Take advantage of features only available in the cloud, such as intelligent security systems, regions footprint, or resiliency features.
- Use the on-demand nature of the cloud to meet peak or seasonal requirements, then releasing that cost investment when it is no longer needed.
- Consolidate physical systems.
- Retire on-premises infrastructure.
- Reduce hardware or data center management costs.
- Increase performance or processing capabilities, through services like big data and machine learning.
- Meet regulatory considerations, including taking advantage of certified infrastructure.

Narrow down each requirement before you start the design of the workload. Expect the requirements to change over time as the solution is deployed and optimized.

Landing zone

Consider the cost implications of the geographic region to which the landing zone is deployed.

The landing zone consists of the subscription and resource group, in which your cloud infrastructure components exist. This zone impacts the overall cost. Consider the tradeoffs. For example, there are additional costs for network ingress and egress for cross-zonal traffic. For more information, see Azure regions and Azure resources.

For information about landing zone for the entire organization, see CAF: Implement landing zone best practices.

Security

Security is one of the most important aspects of any architecture. Security measures protect the valuable data of the organization. It provides confidentiality, integrity, and availability assurances against attacks and misuse of the systems.

Factor in the cost of security controls, such as authentication, MFA, conditional access, information protection, JIT/PIM, and premium Azure AD features. Those options will drive up the cost.

For security considerations, see the Security Pillar.

Business continuity

Does the application have a Service Level Agreement that it must meet?

Factor in the cost when you create high availability and disaster recovery strategies.

Overall Service Level Agreement (SLA), Recovery Time Objective (RTO), and Recovery Point Objective (RPO) may drive towards expensive design choices in order to support higher availability requirements. For example, a choice might be to host the application across regions, which is costlier than single region but supports high availability.

If your service SLAs, RTOs and RPOs allow, then consider cheaper options. For instance, pre-build automation scripts and packages that would redeploy the disaster recovery components of the solution from the ground-up in case a disaster occurs. Alternatively, use Azure platform-managed replication. Both options can lower cost because fewer cloud services are pre-deployed and managed, reducing wastage.

In general, if the cost of high availability exceeds the cost of application downtime, then you could be over engineering the high availability strategy. Conversely, if the cost of high availability is less than the cost of a reasonable period of downtime, you may need to invest more.

Suppose the downtime costs are relatively low, you can save by using recovery from your backup and disaster recovery processes. If the downtime is likely to cost a significant amount per hour, then invest more in the high availability and disaster recovery of the service. It's a three-way tradeoff between cost of service provision, the availability requirements, and the organization's response to risk.

Application lifespan

Does your service run seasonally or follow long-term patterns?

For long-running applications, consider using Azure Reservations if you can commit to one-year or three-year term.

Similarly, Azure savings plans is another option to save money when you have consistent resource usage in your subscriptions and can commit to a fixed hourly spend on compute services for a one-year or three-year period

If your application runs intermittently, consider using consumption billing plans where they are offered on services in your architecture. For example, use Azure Functions in a Consumption plan, so you only pay for compute resources you use.

Automation opportunities

Is it a business requirement to have the service be available 24x7?

You may not have a business goal to leave the service running all the time. Doing so will incur a consistent cost. Can you save by shutting down the service or scaling it down outside normal business hours? If you can,

- Azure has a rich set of APIs, SDKs, and automation technology that utilizes DevOps and traditional automation principles. Those technologies ensure that the workload is available at an appropriate level of scale as needed.
- Repurpose some compute and data resources for other tasks that run out of regular business hours. Reference the Compute Resource Consolidation pattern and consider containers or elastic pools for more compute and data cost flexibility.

Budget for staff education

Keep the technical staff up to date in cloud management skills so that the invested services are optimally used.

- Consider using resources such as FastTrack for Azure and Microsoft Learn training to onboard the staff. Those resources provide engineering investments at no cost to customers.
- Identify training requirements and costs for cloud migration projects, application development, and architecture refinement.
- Invest in key areas, such as identity management, security configuration, systems monitoring, and automation.

- Give the staff access to training and relevant announcements. This way, they can be aware of new cloud capabilities and updates.
- Provide opportunities to get real-world experience of customers across the globe through conferences, specific cloud training, and passing dedicated Microsoft Exams (AZ, MS, MB, etc.).

Standardization

Ensure that your cloud environments are integrated into any IT operations processes. Those operations include user or application access provisioning, incident response, and disaster recovery. That mapping may uncover areas where additional cloud cost is needed.

Next step

Determine the cost constraints

Azure regions

Article • 04/18/2023

The cost of an Azure service can vary between locations based on demand and local infrastructure costs. Consider the following geographical areas when choosing the location of your resources to estimate costs.

Terminology	Description
Azure region	A set of data centers deployed within a latency-defined perimeter and connected through a dedicated regional low-latency network.
Availability zone	A unique physical location within a region with independent power, network, and cooling to be tolerant to data center failures through redundancy and logical isolation of services.
Billing zone	A geographic collection of regions that is used for billing.
Location	A region or a zone within a region. Azure has data centers all over the world and each data center resides in a location.
Landing zone	The ultimate location of your cloud solution or the landing zone. The landing zone typically consists of logical containers like a subscription and resource group, in which your cloud infrastructure components exist.

See the Azure global infrastructure for a complete list of Azure geographies, regions, and locations.

To see product availability by region, see Products available by region.

Tradeoff

- Locating resources in a cheaper region shouldn't negate the cost of network ingress and egress or by degraded application performance because of increased latency.
- Hosting an application in a single region can cost less than an application that's hosted across regions because of replication costs or the need for extra nodes.

Compliance and regulatory

Azure offers differentiated cloud regions for specific security and compliance requirements.

Does your solution need specific levels of security and compliance?

If your solution needs to follow certain government regulations, the cost is higher. Otherwise you can meet less rigid compliance, through Azure Policy, which is free.

Certain Azure regions are built specifically for high compliance and security needs. For example, with Azure Government (USA) you're given an isolated instance of Azure. Azure Germany has data centers that meet privacy certifications. These specialized regions have higher cost.

Regulatory requirements can dictate restrictions on data residency. These requirements can change your data replication options for resiliency and redundancy.

Traffic across billing zones and regions

Cross-regional traffic and cross-zonal traffic incur more costs.

Is the application critical enough to have the footprint of the resources cross zones and,or cross regions?

Bandwidth refers to data moving in and out of Azure data centers. Inbound data transfers, or data transfers going into Azure data centers, are free for most services. For outbound data transfers, the billing zone determines the data transfer pricing. For more information, see Bandwidth Pricing Details.

Suppose you want to build a cost-effective solution by provisioning resources in locations that offer the lowest prices. The dependent resources and their users are located in different parts of the world. In this case, data transfers between locations add cost if there are meters tracking the volume of data moving across locations. The cost of transferring the data could offset any savings from choosing the cheapest location.

- The extra cross-regional and cross-zone costs don't apply to global services, such as Azure Active Directory.
- Not all Azure services support zones and not all regions in Azure support zones.

> Before choosing a location, consider how important is the application to justify the cost of having resources cross zones and/or cross regions. For non-mission critical applications such as, developer or test, consider keeping the solution and its dependencies in a single region or single zone to leverage the advantages of choosing the lower-cost region.

Azure resources

Article • 11/30/2022

Just like on-premises equipment, there are several elements that affect monthly costs when using Azure services.

Usage meters for resources

Most services are priced based on units of size, amount of data, or operations. When you provision an Azure resource, Azure creates metered instances for that resource. The *meters* track the resources' usage and generate a usage record that is used to calculate your bill.

For example, you provision a virtual machine in Azure. Some meters that track its usage include: Compute Hours, IP Address Hours, Data Transfer In, Data Transfer Out, Standard Managed Disk, Standard Managed Disk Operations, Standard IO-Disk, Standard IO-Block Blob Read, Standard IO-Block Blob Write, Standard IO-Block Blob Delete.

How is the usage tracked for each resource in the workload?

For each Azure resource, have a clear understanding of the meters that track usage and the number of meters associated with the resource tier. The meters correlate to several billable units. Those units are charged to the account for each billing period. The rate per billable unit depends on the resource tier.

A resource tier impacts pricing because each tier offers levels of features such as performance or availability. For example, a Standard HDD hard disk is cheaper than a Premium SSD hard disk.

> Start with a lower resource tier then scale the resource up as needed. Growing a service with little to no downtime is easier when compared to downscaling a service. Downscaling usually requires deprovisioning or downtime. In general, choose scaling out instead of scaling up.

As part of the requirements, consider the metrics for each resource and build your alerts on baseline thresholds for each metric. The alerts can be used to fine-tune the resources. For more information, see Respond to cost alert.

Allocated usage for the resource

Another way to look at pricing is the allocated usage.

Suppose, you de-allocate the virtual machine. You'll not be billed for Compute Hours, I/O reads or writes, and other compute meters because the virtual machine is not running and has no given compute resources. However, you'll be charged for storage costs for the disks.

Here are some considerations:

- The meters and pricing vary per product and often have different pricing tiers based on the location, size, or capacity of the resource.
- Cost for associated with infrastructure is kept low with commodity hardware.
- Failures cannot be prevented but the effects of failure can be minimized through design choices. The resiliency features are factored in the price of a service and its features.

Here are some examples:

Azure Disk

Start with a small size in GB for a managed disk instead of pay-per-GB model. It's cost effective because cost is incurred on the allocated storage.

ExpressRoute

Start with a smaller bandwidth circuit and scale up as needed.

Compute infrastructure

Deploy an additional smaller instance of compute alongside a smaller unit in parallel. That approach is more cost effective in comparison to restarting an instance to scale up.

For details about how billing works, see Azure Cost Management + Billing documentation.

Subscription and offer type

What is the subscription and offer type in which resources are created?

Azure usage rates and billing periods can vary depending on the subscription and offer type. Some subscription types also include usage allowances or lower prices. For example, Azure Dev/Test subscription offers lower prices on Azure services such as specific VM sizes, PaaS web apps, and VM images with pre-installed software. Visual Studio subscribers obtain as part of their benefits access to Azure subscriptions with monthly allowances.

For information about the subscription offers, see Microsoft Azure Offer Details.

As you decide on the offer type, consider the types supporting Azure reservations and savings plans.

- With Azure Reservations, you prepay for reserved capacity at a discount. Reservations are suitable for workloads that have a long-term usage pattern. Combining the offer type with reservations can significantly lower the cost. For information about subscription and offer types that are eligible for reservations, see Discounted subscription and offer types.
- With Azure savings plans, you commit to a fixed hourly spend across all compute services and apply to all participating compute services globally up to the hourly commitment.

Billing structure for services in Azure Marketplace

Are you considering third-party services through Azure Marketplace?

Azure Marketplace offers both the Azure products and services from third-party vendors. Different billing structures apply to each of those categories. The billing structures can range from free, pay-as-you-go, one-time purchase fee, or a managed offering with support and licensing monthly costs.

Governance

Article • 04/21/2023

Governance helps with cost management. This work benefits your ongoing cost review process and offers a level of protection for new resources.

Understand how centralized governance functions can support your team

Centralized governance can relieve some of the burden related to on-going cost monitoring and optimization. But it's an augmentation to the workload team's responsibilities, not a replacement. To understand how centralized cloud governance teams operate, see Govern methodology for the cloud.

- For more detailed information on cost optimization, see Cost Management discipline.
- For an example of the types of guardrails provided by a governance team, see Cost management discipline improvement. This article includes examples of suggested tags and policies for improving cost governance.
- If centralized governance teams don't support your team, see Cloud governance function. This article helps you understand the activities your team might consider including in each sprint.

Follow organizational policies that define cost boundaries

Use policies to ensure compliance with identified cost boundaries. Policies eliminate the need for manual resource approval and they speed up provisioning.

Azure Policy can set rules on management groups, subscriptions, and resources groups. The policies control clouds service resource SKU size, replication features, and supported locations. Use policies to prevent provisioning expensive resources. Identify the built-in Azure policies that can help lower costs. For even more control, create custom policies.

For more information, see Create management groups for resource organization and management.

For more information on how to control the group who manages resources in the subscription, see Azure built-in roles.

Set limits or quotas to prevent unexpected costs. For more information, see Azure subscription and service limits, quotas, and constraints.

Enforce resource tagging

Use tags on resources and resource groups to track incurred costs. Identify the service meters that you can't tag or view in the cost analysis tool in the Azure portal.

The advantages of tagging include:

- The cost can be reported to an owner, an application, a business department, or a project initiative. This feature is useful because the overall cost can span multiple resources, locations, and subscriptions.
- Filter information. Use filtering in the Azure portal cost analysis tool to get granular reports.

There are some limitations:

- You can't tag all Azure resources and not all taggable resources in Azure are accounted for in the Azure cost analysis tool.

Estimate the initial cost

Article • 04/20/2023

It's difficult to know your costs before deploying a workload to the cloud. If you use methods for on-premises estimation or you directly map on-premises assets to cloud resources, estimates are inaccurate. For example, if you build your own datacenter, your costs might appear comparable to cloud. Most on-premises estimates don't account for costs like cooling, electricity, IT and facilities labor, security, and disaster recovery.

Here are some best practices:

- Use proof-of-concept deployments to help refine cost estimates.
- Choose the right resources that can handle workload performance. For example, cheaper virtual machines might initially show a lower cost but be more expensive to maintain a certain performance level.
- Accurately reflect the cost associated with right storage type.
- Add hidden costs, such as networking expenses for large data downloads.

Migration workloads

Quantify the cost of running your business in Azure by calculating total cost ownership (TCO) and the return on investment (ROI). Compare those metrics to existing on-premises equivalents.

It's difficult to know costs before migrating to the cloud.

Using on-premises calculation might not accurately reflect the cost of cloud resources. Here are some challenges:

- On-premises TCO might not accurately account for hidden expenses. These expenses include under-utilization of purchased hardware or network maintenance costs like labor and equipment failure.
- Cloud TCO might not accurately account for a drop in the organization's operational labor hours. Cloud provider's infrastructure, platform management services, and other operational efficiencies are included in the cloud service pricing. Especially at a smaller scale, the cloud provider's services don't result in reduction of IT head count.
- ROI might not accurately account for new organizational benefits because of cloud capabilities. It's hard to quantify improved collaboration, reduced time to service customers, and fast scaling with minimal or no downtime.

- ROI might not accurately account for the business process re-engineering required to fully adopt cloud benefits. In some cases, this re-engineering might not occur at all, leaving an organization in a state where they use new technology inefficiently.

Azure provides the following tools to determine cost.

- Microsoft Azure Total Cost of Ownership Calculator to reflect all costs.

  > For migration projects, the TCO Calculator might help, because it populates some common costs but it lets you modify the cost assumptions.

- Azure pricing calculator to assess costs of the services you plan to use in your solution.
- Azure Migrate to evaluate your organization's current workloads in on-premises datacenters. Azure Migrate suggests an Azure replacement solution with virtual machine sizes based on your workload. It also provides a cost estimate.

Example estimate for a microservices workload

Consider this scenario as an example. Use the Azure pricing calculator to estimate the initial cost before the workload deploys. The cost calculates per month or for 730 hours.

In this example, we've chosen the microservices pattern. As the container orchestrator, one of the options could be Azure Kubernetes Service (AKS) that manages a cluster of pods. Choose NGINX ingress controller because it's a well-known controller for such workloads.

> The example uses current prices and is subject to change. The example shows prices and calculations for illustrative purposes only.

Compute

For AKS, there's no charge for cluster management.

For AKS agent nodes, there are many instance sizes and SKUs options. The example workload is expected to follow a long running pattern and you can commit to three years. An instance that's eligible for reserved instances would be a good choice. Lower the cost by choosing the **3-year reserved plan**.

The workload needs two virtual machines. One is for the backend services and the other is for the utility services.

The B12MS instance with two virtual machines is sufficient for this initial estimation. We can lower cost by choosing reserved instances.

Estimated Total: $327.17 per month with upfront payment of $11,778.17.

Application gateway

For this scenario, consider the **Standard_v2 Tier** of Azure Application Gateway because of the autoscaling capabilities and performance benefits. Also, choose consumption-based pricing, which calculates by capacity units (CU). Each capacity unit calculates based on compute, persistent connections, or throughput. For Standard_v2 SKU, each compute unit can handle approximately 50 connections per second with RSA 2048-bit key TLS certificate. For this workload, estimate 10 capacity units.

Estimated Total: $248.64 per month.

Load balancer

NGINX ingress controller deploys a load balancer that routes internet traffic to the ingress. Approximately 15 load balancer rules are needed. NAT rules are free. The main cost driver is the amount of data processed inbound and outbound independent of rules. The estimated traffic is 1 TB (inbound and outbound).

Estimated Total: $96.37 per month.

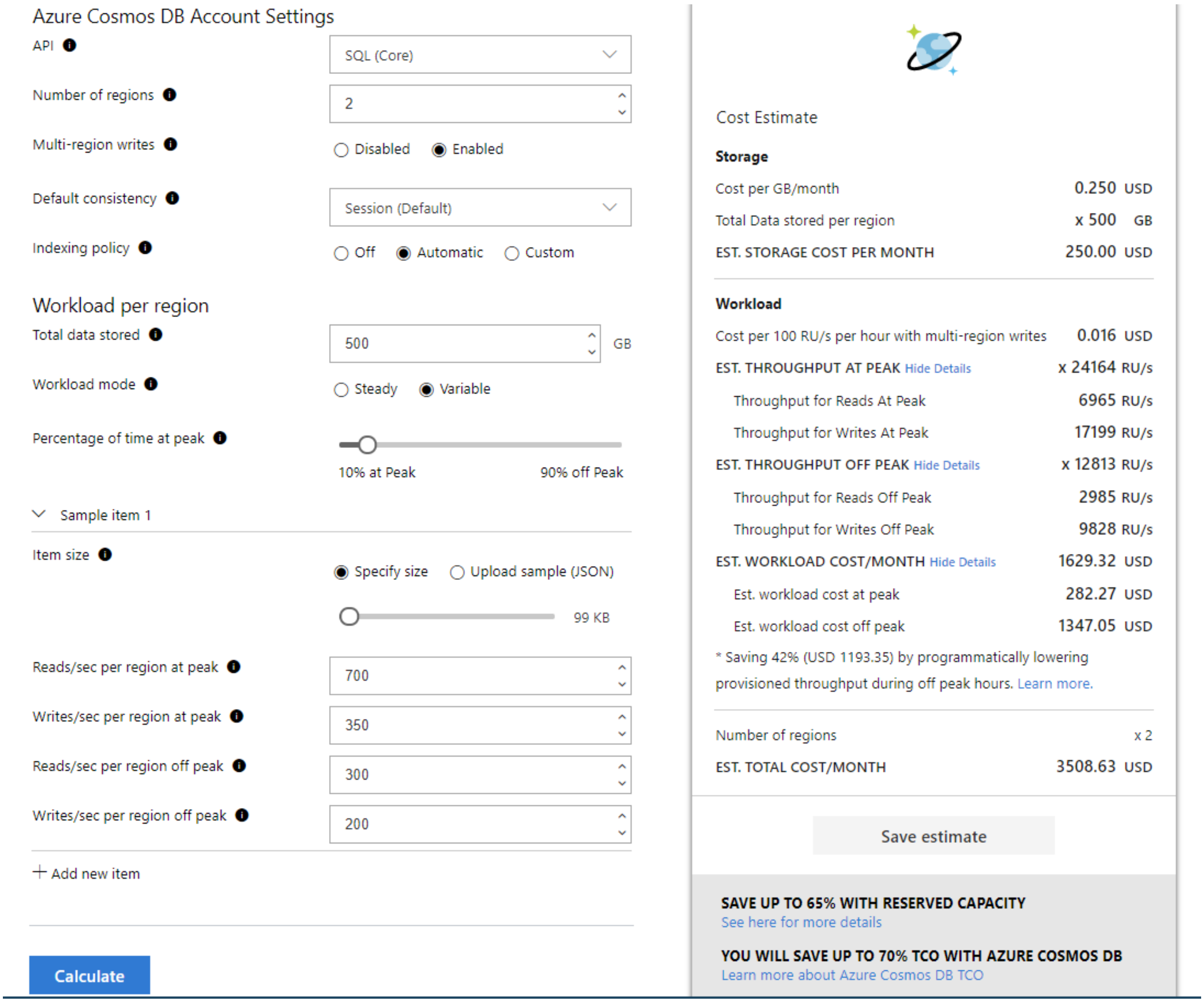
Bandwidth

The estimated outbound traffic is 2 TB. The first 5 GB/month are free in Zone 1, which includes North America, Europe, and Australia. Between 5 GB and 10 TB /month is charged $0.087 per GB.

Estimated Total: $177.74 per month

External data source

Because the schema-on read nature of the data handled by the workload, choose Azure Cosmos DB as the external data store. By using the Azure Cosmos DB capacity calculator, you can calculate the throughput to reserve.

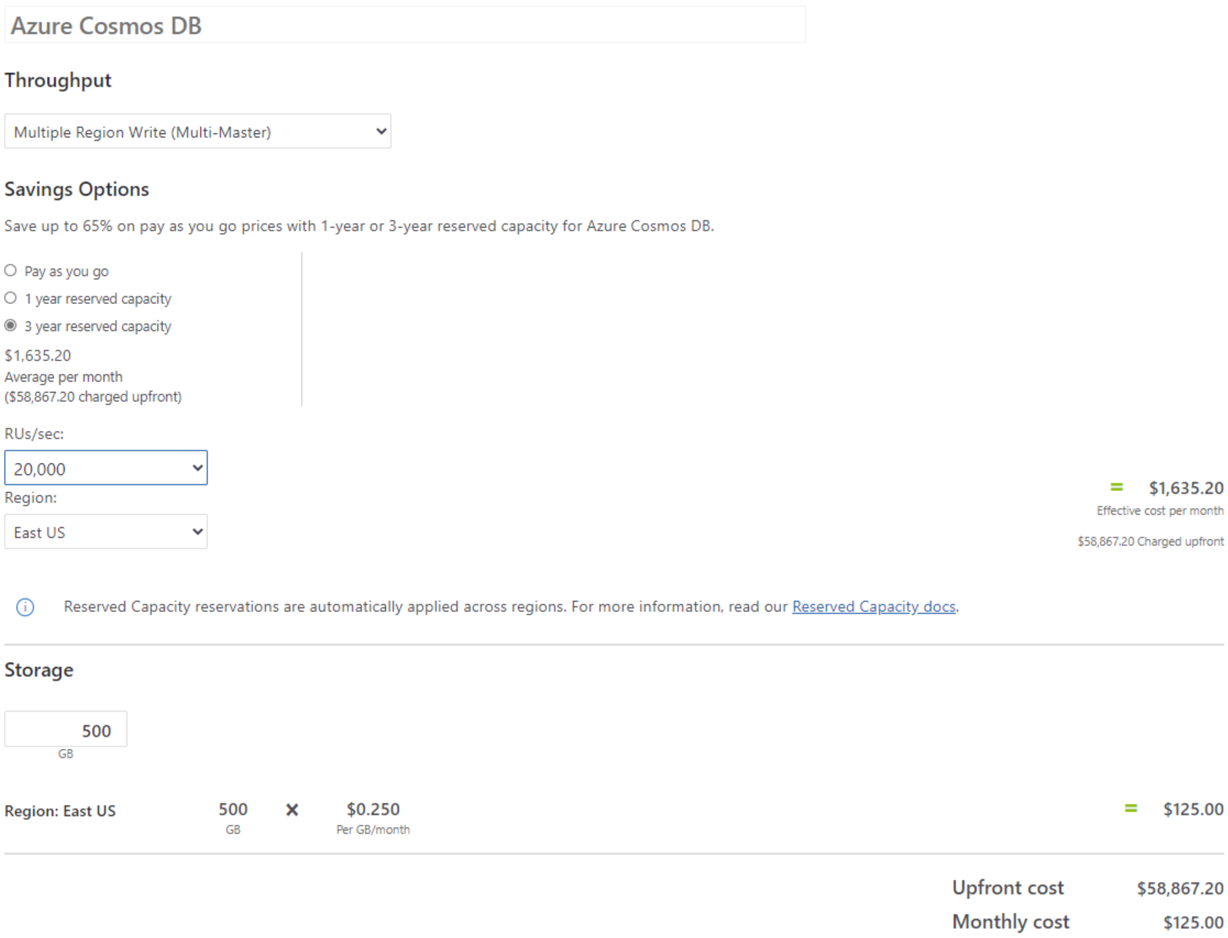


Cost variables

- For lower latency, in this scenario, enable geo-replication by using the **Multi-regions writes** feature. By default, Azure Cosmos DB uses one region for writes and the rest for reads.
- Default choices in consistency model of **Session** and indexing policy to **Automatic**. Automatic indexing makes Azure Cosmos DB indexes all properties in all items for flexible and efficient queries. Custom indexing policy lets you include and exclude properties from the index for lower write request units (RUs) and storage size. Uploading a custom indexing policy can help you reduce costs.
- Total data store isn't a significant cost driver. Here it's set to 500 GB.
- The throughput is variable indicating peaks. The percentage of time at peak is set to 10%.
- The item size is an average of 90k. Using the capacity calculator, you can upload sample json files. The sample json files include your document's data structure, the average document's size, and the number of reads and writes per second. These

variables have the largest effect on cost because you use them to calculate the throughput. The throughput values appear in the image.

Now, use those values in the Azure pricing calculator .



The average throughput based on these settings is 20,000 RUs per second, which is the minimum throughput required for a **3-year reserved capacity** plan.

Here's the total cost for three years using the reserved plan:

$1,635.20 Average per month ($58,867.20 charged upfront)

You save $700.00 by choosing the three-year reserved capacity over the pay-as-you-go price.

CI/CD pipelines

With Azure DevOps, *Basic plan* is included for Visual Studio Enterprise, Professional, Test Professional, and MSDN Platforms subscribers. And there's no charge for adding or editing work items and bugs and viewing dashboards, backlogs, and Kanban boards for stakeholders.

The Basic plan license for five users is free.

More services

For Microsoft Hosted Pipelines, the **Free** tier includes one parallel CI/CD job with 1,800 minutes (30 hours) per month. But you can select the **Paid** tier and have one CI/CD parallel job ($40.00) in this tier. Each parallel CI/CD job includes unlimited minutes.

> For this stage of cost estimation, the self-hosted pipelines isn't required because the workload doesn't have custom software that runs in your build process and which isn't included in the Microsoft-hosted option.

Azure Artifacts is a service you can use to create package feeds to publish and consume Maven, npm, NuGet, Python, and universal packages. Azure Artifacts bills on a consumption basis, and is free up to 2 GB of storage. For this scenario, we estimate 56 GB in artifacts ($56.00)

Azure DevOps offers a cloud-based solution for load testing your apps. Load tests are measured and billed in virtual user minutes (VUMs). For this scenario, estimate a 200,000 VUMs ($72.00).

Estimated Total: $168.00 per month

Managed services

Article • 04/10/2023

Look for areas in the architecture where it makes sense to incorporate platform-as-a-service (PaaS) options. These include caching, queues, and data storage. PaaS reduces time and cost of managing servers, storage, networking, and other application infrastructure.

With PaaS, the infrastructure costs are included in the pricing model of the service. For example, you can allocate a lower SKU virtual machine as a jump box. There are more costs for storage and managing a separate server. You also need to configure a public IP on the virtual machine, which we don't recommend. A managed service, such as Azure Bastion, takes into consideration all those costs and offers better security.

Azure provides a wide range of PaaS resources. Here are some examples of when you might consider PaaS options:

Task	Use
Host a web server	Azure App Service instead of setting up IIS servers.
Indexing and querying heterogenous data	Azure Cognitive Search instead of ElasticSearch.
Host a database server	Azure offers many SQL and no-SQL options such as Azure SQL Database and Azure Cosmos DB.
Secure access to virtual machine	Azure Bastion instead of virtual machines as jump boxes.
Network security	Azure Firewall instead of virtual network appliances.

For more information, see Use platform as a service (PaaS) options.

Reference architecture

To see an implementation that provides better security and lowers cost through PaaS services, go to Network DMZ between Azure and an on-premises datacenter.

Consumption and fixed cost models

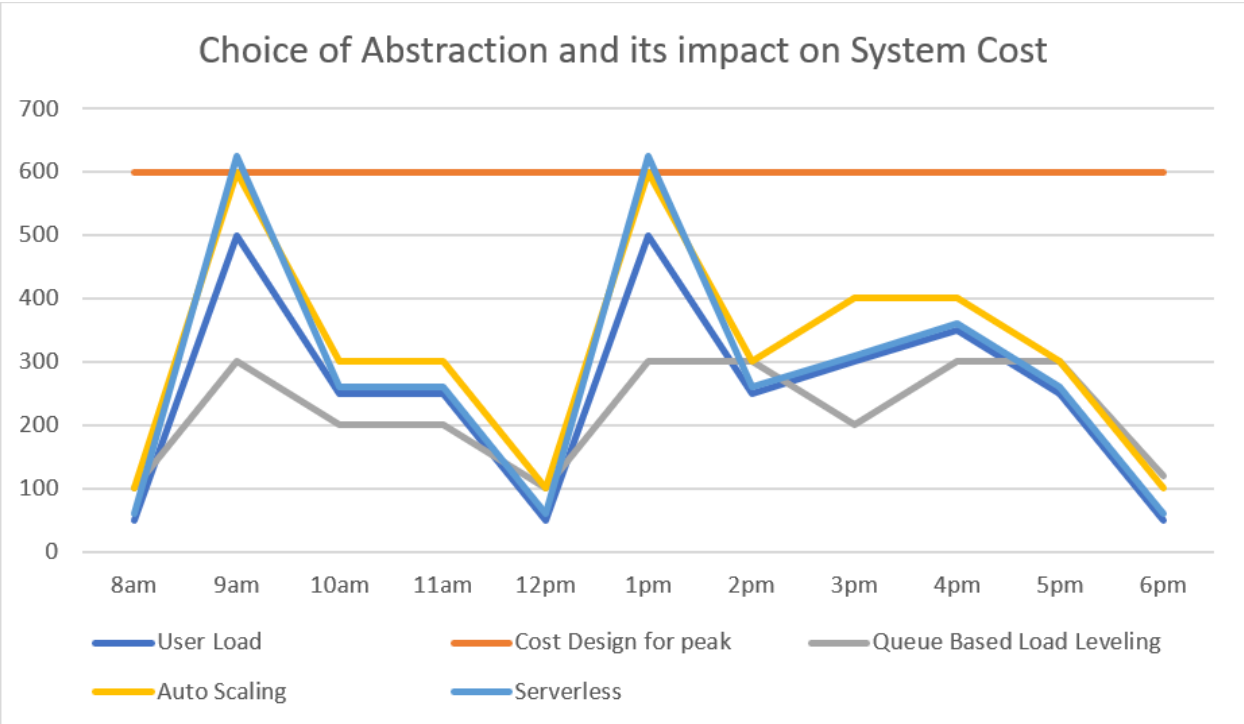
Article • 04/06/2023

The common pricing options for Azure services are:

- Consumption-based price - You're charged for only what you use. This model is also known as the pay-as-you-go rate.
- Fixed price - You provision resources and are charged for those instances whether you use them or not.

A common way to estimate cost is by considering workloads on a peak throughput. With consistently high utilization, consumption-based pricing can be less efficient for estimating baseline costs when compared to the equivalent provisioned pricing. Platform as a service (PaaS) and serverless technologies can help you understand the economy cutoff point for consumption-based pricing.

See the difference between cost models based on fixed, static services provisioning, and more variable costs based on autoscaling serverless technologies.



Start with a fixed minimum-level of performance and then use architectural patterns, such as queue based load leveling and autoscaling services. With this approach, the peaks are smoothed out into a more consistent flow of compute and data. This approach should temporarily extend your burst performance when the service is under sustained load. If cost is an important factor but you need to maintain service availability under burst workloads, use the throttling pattern to maintain service quality under load.

Compare and contrast the options and understand how to provision workloads that can potentially switch between the two models. The model is a tradeoff between scalability and predictability. Ideally in the architecture, you blend the two aspects.

Provisioning cloud resources to optimize cost

Article • 04/18/2023

Deployment of cloud resources of a workload is known as *provisioning*.

Use the Azure pricing calculator to estimate the cost of your SKU choices. This list describes some of your options:

- AI + Machine Learning cost estimates
- Big data analytics cost estimates
- Compute cost estimates
- Networking cost estimates
- Data store cost estimates

AI + Machine Learning cost estimates

Article • 04/20/2023

Compute cost is the main cost driver for machine learning workloads. Those resources are needed to run the training model and host the deployment. For information about choosing a compute target, see What are compute targets in Azure Machine Learning?

The compute cost depends on cluster size, node type, and the number of nodes. Billing occurs while the cluster nodes are starting, running, or shutting down.

With services such as Azure Machine Learning, you can create fixed-sized clusters or use autoscaling.

> If the amount of compute isn't known, start with a zero-node cluster. The cluster scales up when it detects jobs in the queue. There is no charge for a zero-node cluster.

Fix-sized clusters are appropriate for jobs that run at a constant rate and the amount of compute is known and measured beforehand. The time taken to spin up or down a cluster incurs higher cost.

> If you don't need retraining frequently, turn off the cluster when not in use.

To lower the cost for experimental or development workloads, choose Spot VMs. We don't recommend Spot VMs for production workloads because Azure might evict them at any time. For more information, see Use Azure Spot Virtual Machines.

For more information about the services that make up a machine learning workload, see Compare Microsoft machine learning products and technologies.

This article provides cost considerations for some technology choices. This list isn't meant to be exhaustive.

Azure Machine Learning

Training models don't incur the machine learning service surcharge. You're charged for the following factors:

- Compute choices, such as the virtual machine sizes and the region in which they're available, drive costs. If you can commit to one or three years, choosing reserved instances or savings plans can lower cost. For more information, see Reserved VMs and Savings plans.

- As part of provisioning Machine Learning resources, resources are deployed such as Azure Container Registry, Azure Blob Storage, and Key Vault. You incur charges according to the pricing of those individual services.

- If you deploy models to a Kubernetes Service cluster, Machine Learning adds a surcharge on top of the Kubernetes Service compute cost. This cost can be lowered through autoscaling.

For more information, see Pricing calculator

Reference architecture

- Distributed hyperparameter tuning for machine learning models
- Distributed training of deep learning models on Azure
- Batch scoring of Python models on Azure
- Batch scoring for deep learning models using Azure Machine Learning pipelines
- Real-time scoring of machine learning models in Python
- MLOps for Python models using Azure Machine Learning
- Batch scoring with R models to forecast sales
- Real-time scoring of R machine learning models
- Batch scoring of Spark models on Azure Databricks
- Build an enterprise-grade conversational bot
- Build a real-time recommendation API on Azure

Azure Cognitive Services

The billing depends on the type of service. The charges are based on the number of transactions for each type of operation specific to a service. A certain number of transactions are free. If you need more transactions, choose from the *Standard* instances. For more information, see:

- Pricing calculator
- Azure Cognitive Services pricing

Azure Bot Service

The Azure Bot Service is a managed service purpose-built for enterprise-grade bot development. Billing is based on the number of messages. A certain number of messages are free. If you need to create custom channels, choose *Premium channels*, which can drive up the cost of the workload.

For a Web App Bot, an App Service is provisioned to host the bot. Also, an instance of Azure Monitor is provisioned. You're charged per the pricing of those individual services.

Reference architecture for Azure Cognitive Services and Azure Bot Service

- Enterprise-grade conversational bot

Big data analytics cost estimates

Article • 05/25/2023

Most big data workloads are designed to do:

- Batch processing of big data sources at rest.
- Stream processing of data in motion.

Those workloads have different needs. Batch processing is done with long-running batch jobs. For stream processing, the data ingestion component should be able to capture, store, and in some cases buffer real-time messages. Both workloads have the requirement to store a large volume of data then filter, aggregate, and prepare that data for analysis.

For information about choosing technologies for each workload, see:

- Batch processing
  - Technology choices for batch processing
  - Capability matrix
- Stream processing
  - What are your options when choosing a technology for real-time processing?
  - Capability matrix

This article provides cost considerations for some of those choices.

Azure Synapse Analytics

The analytics resources are measured in Data Warehouse Units (DWUs), which tracks CPU, memory, and IO. DWU also indicates the required level of performance. If you need higher performance, add more DWU blocks.

You can provision the resources in one of two service levels.

- **Compute Optimized Gen1** tracks usage in DWUs and is a pay-as-you-go model.
- **Compute Optimized Gen2** tracks the compute DWUs (cDWUs) which lets you scale the compute nodes. This level is intended for intensive workloads with higher query performance and compute scalability. You can choose the pay-as-you-go model or save 37% to 65% by using reserved instances if you can commit to one or three years. For more information, see Reserved VMs.

> Start with smaller DWUs and measure performance for resource intensive operations, such as heavy data loading or transformation. This approach helps you determine the number of units you need to increase or decrease. Measure usage during the peak business hours so that you can assess the number of concurrent queries and accordingly add units to increase the parallelism. Also, measure off-peak usage so that you can pause compute when needed.

In Azure Synapse Analytics, you can import or export data from an external data store, such as Azure Blob Storage and Azure Data Lake Store. Storage and analytics resources aren't included in the price. Moving data in and out of the data warehouse increases bandwidth cost.

For more information, see:

- Azure Synapse Analytics pricing
- Manage compute for dedicated SQL pool in Azure Synapse Analytics

Synapse Analytics reference architecture

- Automated enterprise BI
- Enterprise business intelligence

Azure Databricks

Azure Databricks offers two SKUs, Standard and Premium, each with the following options, in the order of least to most expensive:

- **Data Engineering Light** is for data engineers to build and run jobs in automated Spark clusters.
- **Data Engineering** includes autoscaling and has features for machine learning flows.
- **Data Analytics** includes the preceding features. Data scientists can explore, visualize, manipulate, and share data and insights interactively.

Choose a SKU depending on your workload. If you need features like log audit, which is available in Premium, the overall cost can increase. If you need an autoscaling of clusters to handle larger workloads or interactive Databricks dashboards, choose an option higher than **Data Engineering Light**.

Here are factors that affect Databricks billing:

- Azure location where the resource is provisioned.

- The virtual machine instance tier and the number of hours that the instances run.
- Databricks units (DBU), which is a unit of processing capability per hour, billed on per-second usage.

> This example is based on the current price and is subject to change. The calculation shown is for example purposes only.

The following table provides an example for you running a Premium cluster for 100 hours in East US 2 with 10 DS13v2 instances.

Item	Example estimate
Cost for 10 DS13v2 instances	100 hours x 10 instances x $0.741/hour = $741.00
DBU cost for **Data Analytics** workload	100 hours x 10 instances x 2 DBU per node x $0.55/DBU = $1,100
Total	$1,841

For more information, see Azure Databricks pricing.

If you can commit to one or three years, opt for reserved instances, which can save 38% to 59%. For more information, see Reserved VMs.

> Turn off the Spark cluster when not in use to prevent unnecessary charges.

Databricks reference architecture

- Stream processing with Azure Databricks
- Build a Real-time Recommendation API on Azure
- Batch scoring of Spark models on Azure Databricks

Azure Stream Analytics

Stream analytics uses *streaming units (SUs)* to measure the amount of compute, memory, and throughput required to process data. When you provision a stream processing job, specify an initial number of SUs. Higher streaming units incur a higher cost because more resources are used.

Stream processing with low latency requires a significant amount of memory. The SU% utilization metric tracks this resource. Lower utilization indicates that the workload requires more compute resources. You can set an alert on the **80% SU Utilization** metric to prevent resource exhaustion.

> To evaluate the number of units you need, process an amount of data that's realistic for your production level workload, observe the SU% Utilization metric, and adjust the SU value.

You can create stream processing jobs in Azure Stream Analytics and deploy them to devices running Azure IoT Edge through Azure IoT Hub. The number of devices affects the overall cost. Billing starts when a job is deployed to devices, regardless of the job status (running, failed, stopped).

SUs are based on the partition configuration for the inputs and the query that's defined within the job. For more information, see Calculate the max streaming units for a job and Understand and adjust Stream Analytics streaming units.

For pricing details, see Azure Stream Analytics pricing.

Stream Analytics reference architecture

- Batch scoring of Python models on Azure
- Azure IoT reference architecture

Azure Analysis Services

Big data solutions need to store data that you can use for reporting. Azure Analysis Services supports the creation of tabular models to meet this need.

Here are the tiers:

- **Developer** is recommended for evaluation, development, and test scenarios.
- **Basic** is recommended for small production environment.
- **Standard** is recommended for mission critical workloads.

Analysis Services uses Query Processing Units (QPUs) to determine the processing power. A QPU is an abstracted measure of compute and data processing resources that affect performance. A higher QPU results in higher performance.

Each tier offers one or more instances. The main cost drivers are the QPUs and memory allocated for the tier instance. Start with a smaller instance, monitor the QPU usage, and scale up or down by selecting a higher or lower instance within the tier. Also, monitor usage during off-peak hours. You can pause the server when not in use. No charges apply when you pause your instance. For more information, see:

- The right tier when you need it
- Monitor server metrics
- Azure Analysis Services pricing

Analysis Services reference architecture

- Enterprise business intelligence
- Automated enterprise BI

Azure Data Factory V2

Azure Data Factory is a big data orchestrator. The service transfers data to and from diverse types of data stores. It transforms the data by using other compute services. It creates workflows that automate data movement and transformation. You're only charged for consumption. The following factors measure consumption:

- Pipeline activities that take the actions on the data. Those actions include copying the data from various sources, transforming it, and controlling the flow. For more information, see:
  - Copy activity in Azure Data Factory and Azure Synapse Analytics
  - Transform data in Azure Data Factory and Azure Synapse Analytics
  - Web activity in Azure Data Factory and Azure Synapse Analytics.

- Actions measured in data integration units. Each unit tracks CPU, memory, and network resource allocation. This measurement applies to Azure Integration Runtime.

You're charged for the total activities in thousands. You're also charged for activities, such as copying data, lookups, and external activities. Each activity is individually priced. You're also charged for pipelines with no associated triggers or runs within the month. All activities are prorated by the minute and rounded up.

Azure Data Factory V2 reference architecture

- Automated enterprise BI
- Enterprise business intelligence
- Build an enterprise-grade conversational bot

Compute cost estimates

Article • 04/20/2023

Compute refers to the hosting model for the computing resources that your application runs on:

- *Infrastructure as a service* (IaaS) lets you provision individual virtual machines (VMs) along with the associated networking and storage components. Then you deploy whatever software and applications you want onto those VMs. This model is the closest to a traditional on-premises environment, except that Microsoft manages the infrastructure. You still manage the individual VMs.

- *Platform as a service* (PaaS) provides a managed hosting environment, where you can deploy your application without needing to manage VMs or networking resources. Azure App Service is a PaaS service.

- *Functions as a service* (FaaS) goes even further in removing the need to worry about the hosting environment. In a FaaS model, you deploy your code and the service automatically runs it. Azure Functions is a FaaS service.

Hosting models

Whether you're hosting model is IaaS, PaaS, or FaaS, each resource requires you to evaluate the tradeoffs that can affect your cost. To learn more about hosting models, see Understand the hosting models.

What are the cost implications of choosing a hosting model?

If your application can be broken down into short pieces of code, a FaaS hosting model might be the best choice. You're charged only for the time it takes to run your code. For example, Azure Functions is a FaaS service that processes events with serverless code. Azure Functions lets you run short pieces of code, called *functions*, without worrying about application infrastructure. Use one of the three Azure Functions pricing plans to fit your need. For more information, see Introduction to Azure Functions.

If you want to deploy a larger or more complex application, PaaS might be the better choice. With PaaS, your application is always running, as opposed to FaaS, where your code runs only when needed. Since PaaS uses more resources, the cost increases.

If you're migrating your infrastructure from on-premises to Azure, IaaS greatly reduces and optimizes infrastructure costs and salaries for IT staff who are no longer needed to manage the infrastructure. Since IaaS uses more resources than PaaS and FaaS, your cost could be highest.

What are the main cost drivers for Azure services?

You're charged differently for each service depending on your region, licensing plan (such as, Azure Hybrid Benefit for Windows Server), number and type of instances you need, operating system, lifespan, and other parameters required by the service. Assess the need for each compute service by using the flowchart in Choose a candidate service.

Consider the tradeoffs that affect your cost by creating different estimates using the Pricing Calculator. If your application consists of multiple workloads, we recommend that you evaluate each workload separately. Refer to Consider limits and costs to do a more detailed evaluation on service limits, cost, SLAs, and regional availability.

Are there payment options for Virtual Machines (VMs) to help meet my budget?

The business requirements of your workload drive your best choice. If you have higher SLA requirements, those requirements increase overall costs. You might need more VMs to ensure uptime and connectivity. Other factors that affect cost are region, operating system, and the number of instances you choose. Your cost also depends on the workload life span. For more information, see Virtual machines and Use Azure Spot Virtual Machines.

- *Pay as you go* lets you pay for compute capacity by the second, with no long-term commitment or upfront payments. This option lets you increase or decrease compute capacity on demand. It's appropriate for applications with short-term, spiky, or unpredictable workloads that can't be interrupted. You can also start or stop usage at any time, resulting in paying only for what you use.

- *Reserved Virtual Machine Instances* lets you purchase a VM for one or three years in a specified region in advance. It's appropriate for applications with steady-state usage. You might save more money compared to pay-as-you-go pricing.

- *Savings plans* let you commit to one or three years, across all regions, for a fixed, hourly dollar amount across compute services.

- *Spot pricing* lets you purchase unused compute capacity at major discounts. If your workload can tolerate interruptions, and the run time is flexible, using spot pricing for VMs can significantly reduce the cost of running your workload in Azure.

- *Dev-Test pricing* offers discounted rates on Azure to support your ongoing development and testing. Dev-Test lets you quickly create consistent development and test environments through a scalable, on-demand infrastructure. This approach allows you to spin up what you need, when you need it, and explore scenarios before going into production. To learn more about Azure Dev-Test reduced rates, see Azure Dev/Test Pricing .

For details on available sizes and options for the Azure VMs that you can use to run your apps and workloads, see Sizes for virtual machines in Azure. For details on specific Azure VM types, see Virtual Machine series .

Do I pay extra to run large-scale parallel and high-performance computing batch jobs?

Use Azure Batch to run large-scale parallel and high-performance computing (HPC) batch jobs in Azure. You can save on VM cost because multiple apps can run on one VM. Configure your workload with either the Low-priority tier, which is the cheapest option, or the Standard tier, which provides better CPU performance. There's no cost for the Azure Batch service. Charges accrue for the underlying resources that run your batch workloads.

Use PaaS as an alternative to buying VMs

When you use the IaaS model, you have final control over the VMs. It might appear to be a cheaper option at first, but when you add operational and maintenance costs, the cost increases. When you use the PaaS model, these extra costs are included in the pricing. In some cases, PaaS services can be a cheaper than managing VMs on your own. For help with finding areas in the architecture where it might be natural to incorporate PaaS options, see Managed services.

How can I cut costs with hosting my web apps in PaaS?

If you host your web apps in PaaS, choose an App Service plan to run your apps. The plan defines the set of compute resources on which your app runs. If you have more than one app, they run using the same resources. This situation is where you see the most significant cost savings, because you don't incur cost for VMs.

If your apps are event-driven with a short-lived process using a microservices architecture style, we recommend using Azure Functions. Run time and memory for a single function run determine your cost. For pricing details, see Azure Functions pricing.

Is it more cost-effective to deploy development testing on a PaaS or IaaS hosting model?

If your development testing (dev-test) is built on Azure managed services such as Azure DevOps, Azure SQL Database, Azure Cache for Redis, and Application Insights, the cheapest solution might be using the PaaS hosting model. You don't incur the cost and maintenance of hardware. If your dev-test is built on Azure managed services such as Azure DevOps, Azure DevTest Labs, VMs, and Application Insights, you need to add the cost of the VMs. This cost can greatly increase your cost. For details on evaluating, see Azure Dev/Test Pricing.

A special case of PaaS - Containers

How can I get the best cost savings for a containerized workload that requires full orchestration?

Your business requirements might require that you store container images to provide fast, scalable retrieval, and network-close container workload deployments. Although there are choices as to how you run them, we recommend that you use Azure Kubernetes Service (AKS) to set up instances with a minimum of three nodes. AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure. There's no charge for AKS cluster management. Any other costs are minimal. The containers themselves have no effect on cost. You pay only for per-second billing and custom machine sizes.

Can I save money if my containerized workload doesn't need full orchestration?

Your business requirements might not need full orchestration. If not, and you're using App Service containers, we recommend that you use one of the App Service plans. Choose the appropriate plan based on your environment and workload.

There's no charge to use SNI-based TLS/SSL. Standard and Premium service plans include the right to use one IP TLS/SSL at no extra charge. Free and shared service plans don't support TLS/SSL. You can purchase the right to use more TLS/SSL connections. In all cases, you must purchase the TLS/SSL certificate itself separately. To learn more about each plan, see App Services pricing.

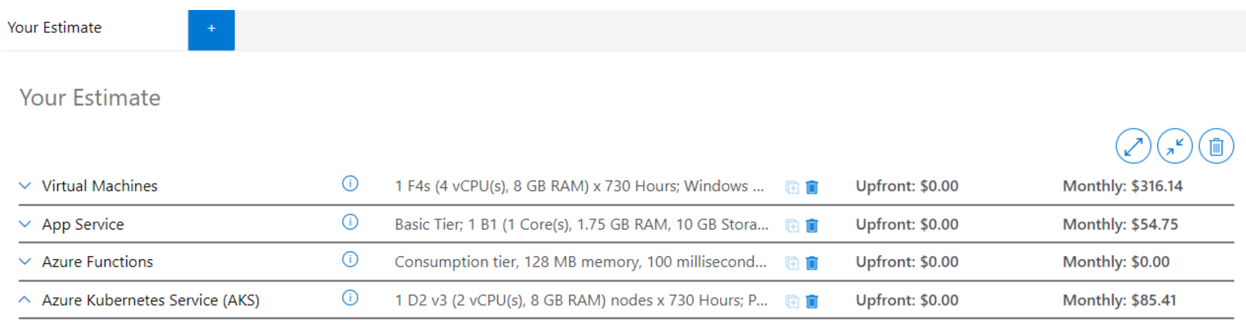
Where's the savings if my workload is event driven with a short-lived process?

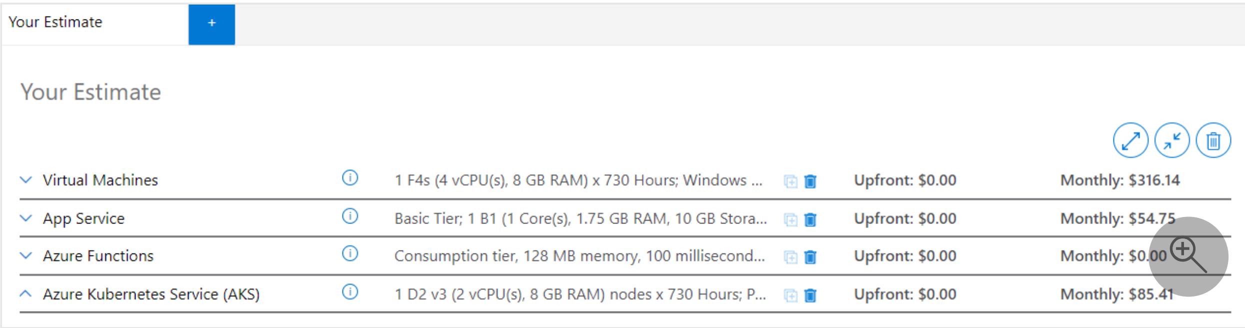
In this example, Service Fabric might be a better choice than AKS. The biggest difference between the two is that AKS works only with Docker applications using Kubernetes. Service Fabric works with microservices and supports many different runtime strategies. Service Fabric can deploy Docker and Windows Server containers. Like AKS, Service Fabric is built for the microservice and event-driven architectures. AKS is strictly a service orchestrator and handles deployments, whereas Service Fabric also offers a development framework that lets you build stateful/stateless applications.

Predict cost estimates using the pricing calculator

Use the Pricing Calculator to create your total cost estimate. After you run your initial scenario, you might find that your plan is beyond the scope of your budget. You can adjust the overall cost and create various cost scenarios to make sure your needs are met before you commit to purchasing.

> **Note**
>
> The costs in this example are based on the current price and are subject to change. The calculation is for illustration purposes only. It shows the Collapsed view of the cost in this estimate.





Tip

You can start building your cost estimate at any time and re-visit it later. The changes are saved until you modify or delete your estimate.

Next steps

- Provisioning cloud resources to optimize cost
- Virtual machines in Azure
- Virtual machines pricing
- Pricing calculator

Data store cost estimates

Article • 04/20/2023

Most cloud workloads adopt the *polyglot* persistence approach. Cloud workloads use a mix of technologies instead of one data store service. You can achieve optimal cost benefit from using this approach.

Each Azure data store has a different billing model. To establish a total cost estimate:

- Identify the business transactions and their requirements
- Break each transaction into operations
- Identify a data store appropriate for the type of data

Perform these steps for each workload separately.

For example, take an e-commerce application. It needs to store data for transactions such as orders, payments, and billing. The data structure is predetermined and not expected to change frequently. Data integrity and consistency are crucial. There's a need to store product catalogs, social media posts, and product reviews. In some cases, the data is unstructured, and is likely to change over time. Media files must be stored and data must be stored for auditing purposes.

Learn about data stores in Understand data store models.

Identify the business transactions and their requirements

The following list of questions address the requirements that can have the greatest effect on your cost estimate. For example, your monthly bill might be within your budget now, but if you scale up or add storage space later, your cost might increase well over your budget.

- Does your data need to be migrated to on-premises, external data centers, or other cloud-hosting environments?
- What type of data do you intend to store?
- How large are the entities you need to store?
- What is the overall amount of storage capacity you need?
- What kind of schemas apply to your data, for example, fixed schema, schema-on-write, or schema-on-read?
- What are your data performance requirements? For example, what are acceptable response times for querying and aggregation of data once ingested?

- What level of fault-tolerance do you need to provide for data consumers?
- What kind of data replication capabilities do you require?
- Do the limits of a particular data store support your requirements for scale, number of connections, and throughput?
- How many instances need to run to support your uptime and throughput requirements? Consider operations costs in this calculation.
- Can you partition your data to store it more cost effectively? For example, can you move large objects out of an expensive relational database into an object store?

There are other requirements that might not have as great of an effect on your cost. For example, the US East region is only slightly lower in cost than Canada Central. See Criteria for choosing a data store for other business requirements.

Use the Azure pricing calculator to determine different cost scenarios.

What are the network requirements that might affect cost?

- Do you need to restrict or otherwise manage access to your data from other network resources?
- Does data need to be accessible only from inside the Azure environment?
- Does the data need to be accessible from specific IP addresses or subnets?
- Does the data need to be accessible from applications or services hosted on-premises or in other external data centers?

Break each transaction into operations

For example, these distinct operations might process one business transaction:

- 10-15 database calls
- Three append operations on blob
- Two list operations on two separate file shares

How does data type in each operation affect cost?

- Consider Azure Blob Storage Block Blobs instead of storing binary image data in Azure SQL Database. Blob storage is cheaper than Azure SQL Database.
- If your design requires SQL, store a lookup table in SQL Database and retrieve the document when needed to serve it to the user in your application middle tier. SQL Database is highly targeted for high-speed data lookups and set-based operations.

- The hot access tier of Azure Block Blob Storage cost is cheaper than the equivalent size of the Premium SSD volume that has the database.

For more information, see Understand data store models.

Identify a data store appropriate for the type of data

> **Note**
>
> An inappropriate data store or one that's mis-configured can have a huge cost impact on your design.

Relational database management systems cost

When you need strong consistency guarantees, we recommend relational database management systems (RDBMS). An RDBMS typically supports a schema-on-write model, where the data structure is defined ahead of time, and all read or write operations must use the schema.

How can I save money if my data is on-premises and already on SQL server?

The on-premises license with Software Assurance can be used to bring down the cost if the workload is eligible for Azure Hybrid Benefit. This option applies for Azure SQL Database (PaaS) and SQL Server on Azure Virtual Machines (IaaS).

For open-source databases such as MySQL, MariaDB, or PostGreSQL, Azure provides managed services that are easy to provision.

What are some design considerations that affect cost?

- If the SLAs don't allow for downtime, can a read-only replica in a different region enable business continuity?
- If the database in the other region must be read/write, how is the data replicated?
- Does the data need to be synchronous, or could consistency allow for asynchronous replication?
- How important is it for updates made in one node to appear in other nodes, before further changes can be made?

Azure storage has several options to make sure data is copied and available when needed. Locally redundant storage (LRS) synchronously replicates data in the primary region. If the entire primary center is unavailable, the replicated data is lost. At the expensive end, Geo-zone-redundant storage (GZRS) replicates data in availability zones within the primary region and asynchronously copied to another paired region. Databases that offer geo-redundant storage, such as SQL Database, are more expensive. Most other OSS RDBMS databases use LRS storage, which contributes to the lower price range.

For more information, see Automated backups in Azure SQL Database.

Azure Cosmos DB offers five consistency levels: *strong*, *bounded staleness*, *session*, *consistent prefix*, and *eventual*. Each level provides availability and performance tradeoffs. Comprehensive SLAs back each level. The consistency level itself doesn't affect cost.

How can I minimize compute cost?

Higher throughput and IOPS require higher compute, memory, I/O, and storage limits. These limits are expressed in a vCore model. With higher vCore number, you buy more resources, so the cost is higher. Azure SQL Database has more vCores and allows you to scale in smaller increments. Azure Database for MySQL, PostgreSQL, and MariaDB have fewer vCores and scaling up to a higher vCore can cost more. MySQL provides in-memory tables in the *Memory Optimized* tier, which can also increase the cost.

All options offer a consumption and provisioned pricing models. With preprovisioned instances, you save more if you can commit to one or three years. To learn more, see Azure Reserved Capacity for Databases.

How is primary and backup storage cost calculated?

With Azure SQL Database, the initial 32 GB of storage is included in the price. For the other listed options, you need to buy storage separately and might increase the cost depending on your storage needs.

For most databases, there's no charge for the price of backup storage that's equal in size to primary storage. If you need more backup storage, you incur more cost.

Key/value and document databases cost

We recommend Azure Cosmos DB for key/value stores and document databases. Azure Cache for Redis is also recommended for key/value stores.

For Azure Cosmos DB, here are some considerations that affect cost:

- Can storage and item size be adjusted?
- Can index policies be reduced or customized to bring down the extra cost for writes and eliminate the requirement for more throughput capacity?
- If data is no longer needed, can it be deleted from the Azure Cosmos DB account? As an alternative, you can migrate the old data to another data store such as Azure blob storage or Azure data warehouse.

For Azure Cache for Redis, there's no upfront cost, no termination fees, you pay only for what you use, and billing is per-hour.

Graphic databases cost

Cost considerations for graphic database stores include storage required by data and indexes used across all the regions. For example, graphs need to scale beyond the capacity of a single server, and make it easy to look up the index when processing a query.

The Azure service that supports this functionality is Azure Cosmos DB for Apache Gremlin. Cost is limited to the Azure Cosmos DB usage. You pay for the operations that you perform on the database and for the storage consumed by your data. Charges are determined by the number of provisioned containers, the number of hours the containers were online, and the provisioned throughput for each container.

If the on-premises data is already on an SQL server on Azure Virtual Machines (IaaS), the license with Software Assurance can be used to bring down the cost if the workload is eligible for Azure Hybrid Benefit.

Data analytics cost

Cost considerations for data analytics stores include data storage, multiple servers to maximize scalability, and the ability to access large data as external tables.

Azure has many services that support data analytics stores:

- Azure Synapse Analytics
- Azure Data Lake
- Azure Data Explorer

- Azure Analysis Services
- Azure HDInsight
- Azure Databricks

As an example of use, historical data is typically stored in data stores such as blob storage or Azure Data Lake Storage Gen2. Azure Synapse, Databricks, or HDInsight access these stores as external tables.

When you use Azure Synapse, you only pay for the capabilities that you opt in to use. During public preview, there's no cost for provisioning an Azure Synapse workspace. Enabling a managed virtual network and Customer Managed Keys might incur a workspace fee after public preview. Pricing of workspaces with other capabilities will be announced at a future date.

Column family database cost

The main cost consideration for a column family database is that it needs to be massively scalable.

The Azure services that support column family databases are Azure Cosmos DB for Apache Cassandra and HBase in HDInsight.

For Azure Cosmos DB for Apache Cassandra, you pay the cost of Azure Cosmos DB, which includes database operations and consumed storage. Azure Cosmos DB for Apache Cassandra is open-source.

For HBase in HDInsight, you pay the cost of HDInsight, which includes instance size and number. HBase is open-source.

Search engine database cost

Cost incurs for a search engine database when applications need to search for information held in external data stores. It also needs to index massive volumes of data and provide near real-time access to these indexes.

Azure Cognitive Search is a search service that uses AI capabilities to identify and explore relevant content at scale.

Time series database cost

The main cost considerations for a time series database is the need to collect large amounts of data in real time from a large number of sources. Although the records written to a time series database are generally small, there are often a large number of records, and total data size can grow rapidly, which drives up cost.

Azure Time Series Insights might be the best option to minimize cost.

Object storage cost

Cost considerations include storing large binary objects such as:

- Images
- Text files
- Video and audio streams
- Large application data objects and documents
- Virtual machine disk images

Azure services that support object storage are Azure Blob Storage and Azure Data Lake Storage Gen2 .

Shared files cost

The main cost consideration is having the ability to access files across a network. For example, given appropriate security and concurrent access control mechanisms, sharing data in this way can enable distributed services to provide highly scalable data access for performing basic, low-level operations such as simple read and write requests.

The Azure service that supports shared files is Azure Files .

Azure messaging cost estimates

Article • 04/20/2023

The messaging services in this article have no up-front cost or termination fees. You pay only for what you use. In some cases, combining two messaging services to increase the efficiency of your messaging system is advantageous. For for more information, see Crossover scenarios.

Cost is based on the number of operations or throughput units you use, depending on the message service. If you use the wrong messaging service, you may incur higher costs. Before choosing a service, first, determine the intent and requirements of the messages. Then consider the tradeoffs between cost, operations units, and throughput units. For tradeoff examples, see Technology choices for a message broker.

Use the Azure pricing calculator for help with creating various cost scenarios.

Azure Service Bus cost

Connect on-premises and cloud-based applications and services to implement highly secure messaging workflows by using Azure Service Bus. Cost is based on messaging operations and the number of connections. The basic tier is the cheapest. If you want more operations and features, choose the standard tier or the premium tier. For example, the Azure Service Bus premium tier runs in dedicated resources to provide higher throughput and more consistent performance.

For pricing details, see Azure Service Bus pricing.

Event Grid cost

Use Event Grid to simplify event-based app development and manage events routing from any source to any destination. Event Grid can route a massive number of events, per second, per region. Cost is based on the number of operations you run. Examples of operations you can run are event ingress, subscription delivery attempts, management calls, and filtering by subject suffix.

For pricing details, see Event Grid pricing.

Event Hubs

Use Event Hubs to stream millions of events per second from any source, build dynamic data pipelines, and immediately respond to business challenges. Cost is based on throughput units. A key difference between Event Grid and Event Hubs is in the way event data is made available to subscribers. For more information, see Pull model.

For frequently asked questions on pricing, see Event Hubs FAQs For details on Event Hubs pricing, see Event Hubs pricing .

Networking cost estimates

Article • 04/20/2023

Traffic routing and load balancing

Most workloads have a load balancing service to route and distribute traffic. It's done in such a way that a single resource isn't overloaded and the response time is minimum with maximum throughput.

Do you need to distribute traffic within a region or across regions?

Azure Front Door and Azure Traffic Manager can distribute traffic to backends, clouds, or hybrid on-premises services that reside in multiple regions. Otherwise, for regional traffic that moves within virtual networks or zonal and zone-redundant service endpoints within a region, choose Azure Application Gateway or Azure Load Balancer.

What is the type of traffic?

Load balancers such as Azure Application Gateway and Azure Front Door, route and distribute traffic to web applications or HTTP/HTTPS endpoints. Those services support Layer 7 features such as TLS/SSL offload, web application firewall, path-based load balancing, and session affinity. For nonweb traffic, choose Azure Traffic Manager or Azure Load Balancer. Azure Traffic Manager uses DNS to route traffic. Load Balancer supports Layer 4 features for all UDP and TCP protocols.

Here's the matrix for choosing load balancers by considering both dimensions.

Service	Global/regional	Recommended traffic
Azure Front Door	Global	HTTP/HTTPS
Azure Traffic Manager	Global	non-HTTP/HTTPS
Azure Application Gateway	Regional	HTTP/HTTPS
Azure Load Balancer	Regional	non-HTTP/HTTPS

For more information, see Choose a load balancing service.

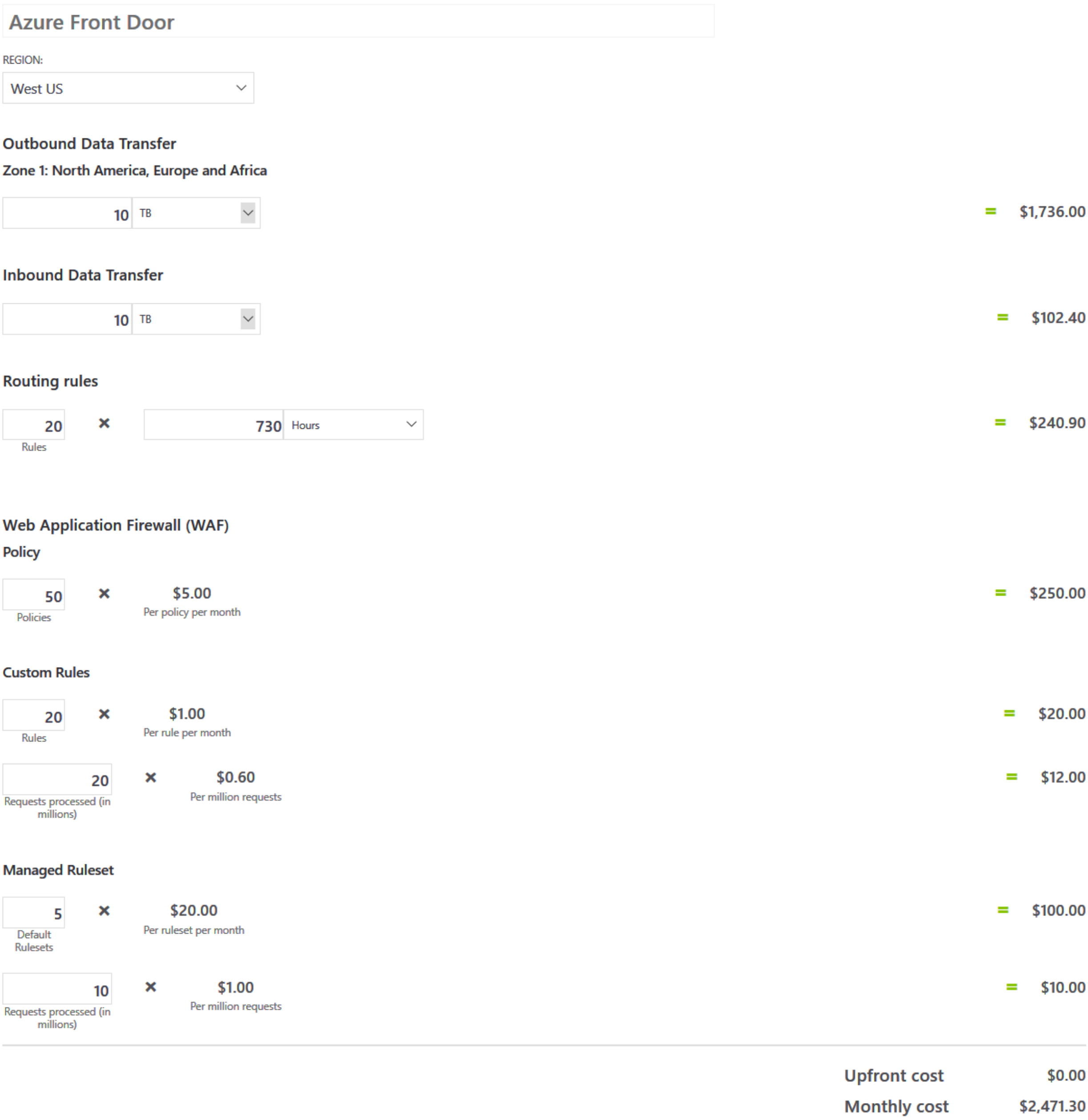
Example cost analysis

Consider a web application that receives traffic from users across regions over the internet. To minimize the request response time, the load balancer can delegate the responsibility from the web server to a different computer. The application is expected to consume 10 TB of data. Routing rules direct incoming requests to various paths. Also, use Web Application Firewall (WAF) to secure the application through policies.

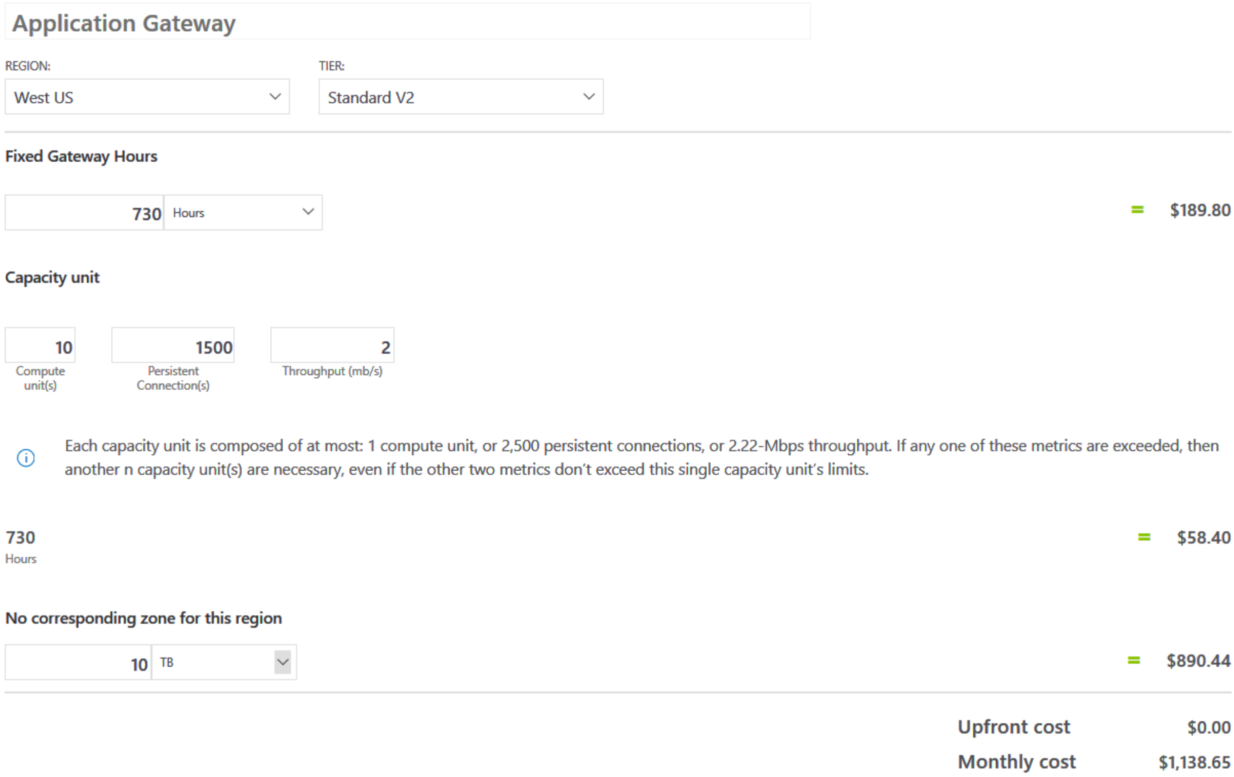
By using the [Azure pricing calculator](#), you can estimate the cost for these two services.

> This example uses current prices and is subject to change. The example shows prices and calculations for illustrative purposes only.

Azure Front Door (West US), Zone 1



Azure Application Gateway (West US)



Consider a similar example where the type of traffic changes. The application is a UDP streaming service that deploys across regions and the traffic goes over the internet. Use a combination of Azure Traffic Manager and Azure Load Balancer. Azure Traffic Manager is a simple routing service that uses DNS to direct clients to a specific service. Here are the cost estimates:

Azure Traffic Manager

Item	Example estimate
DNS queries	100 million x $0.54/per million = $54.00
Basic health checks	20 endpoints x $0.36/month = $7.20
Fast interval health checks	10 endpoints x $1.00/month = $10.00
External endpoints hosted outside Azure	10 endpoints x $0.54/month = $5.40
Real user measurements	100 million user measurements x $2.00/month = $200.00
Traffic view	100 million data points processed x $2.00/month = $200.00
Total	$486.60

Azure Load Balancer

Item	Example estimate
Rules	First five rules: $0.032/rule/hour x 730 = $18.25
	Extra five rules: $0.013/rule/hour x 730 = $36.50
Data processed	10 TB x $0.005/GB = 51.20
Total	$105.95

Peering

Do you need to connect virtual networks?

Peering technology is a service that you use for Azure virtual networks to connect with other virtual networks in the same or different Azure region. Hub and spoke architectures often use peering technology.

There are extra costs incurred by peering connections on both egress and ingress traffic traversing the peering connections.

> If possible, keep the top talking services of a workload within the same virtual network, zone, and region. Use virtual networks as shared resources for multiple workloads against a single virtual network per workload approach. This approach localizes traffic to a single virtual network and avoids the additional costs on peering charges.

Example cost analysis

> This example uses current prices and is subject to change. The example shows prices and calculations for illustrative purposes only.

Peering within the same region is cheaper than peering between regions or Global regions. For instance, you consume 50 TB per month by connecting two VNETs in Central US. Using the current price here's the incurred cost.

Item	Example estimate
Ingress traffic	50 TB x $0.0100/GB = $512.50
Egress traffic	50 TB x $0.0100/GB = $512.50
Total	$1,025.00

Compare that cost for cross-region peering between Central US and East US.

Item	Example estimate
Ingress traffic	50 TB x $0.0350/GB = $1,792.00
Egress traffic	50 TB x $0.0350/GB = $1,792.00
Total	$3,584.00

Hybrid connectivity

There are two main options for connecting an on-premises datacenter to Azure datacenters:

- Azure VPN gateway. Use Azure VPN gateway to connect a virtual network to an on-premises network through a VPN appliance or to Azure Stack through a site-to-site VPN tunnel.
- Azure ExpressRoute. Create private connections between Azure datacenters and infrastructure that's on premise or in a colocation environment.

What is the required throughput for cross-premises connectivity?

We recommend VPN gateway for development and test cases or small-scale production workloads where throughput is less than 100 Mbps. Use Azure ExpressRoute for enterprise and mission-critical workloads that access most Azure services. You can choose bandwidth from 50 Mbps to 10 Gbps.

Another consideration is security. Unlike VPN gateway traffic, Azure ExpressRoute connections don't go over the public internet. Industry-standard IPsec secures the VPN gateway traffic.

For both services, inbound transfers are free and outbound transfers are billed per the billing zone.

For more information, see Choose a solution for connecting an on-premises network to Azure.

This article provides a comparison of the two services.

Example cost analysis

This example compares the pricing details for VPN gateway and Azure ExpressRoute.

> The example uses current prices and is subject to change. The example shows prices and calculations for illustrative purposes only.

Azure VPN Gateway

Suppose you choose the **VpnGw2AZ** tier, which supports availability zones, 1 GB/s bandwidth. The workload needs 15 site-to-site tunnels and 1 TB of outbound transfer. The gateway is provisioned and available for 720 hours.

Item	Example estimate
VPN Hours	720 x $0.564 = $406.08
Site to Site (S2S) Tunnels	The first 10 tunnels are free. For more tunnels, the cost is five tunnels x 720 hours x $0.015 per hour per tunnel = $54.00
Outbound transfer	1024 GB x 0.086 = $88.65
Total cost per month	$548.73

Azure ExpressRoute

Choose the **Metered Data** plan in billing zone of **Zone 1**.

Item	Example estimate
Circuit bandwidth	1 GB/s speed has a fixed rate of $436.00 for the standard price.
Outbound transfer	1 TB x $0.0025 = $25.60
Total cost per month	$461.00

The main cost driver is outbound data transfer. Azure ExpressRoute is more cost-effective than Azure VPN Gateway when consuming large amounts of outbound data. If you consume more than 200 TB per month, consider Azure ExpressRoute with the **Unlimited Data** plan where you're charged a flat rate.

For pricing details, see:

- Azure VPN Gateway pricing
- Azure ExpressRoute pricing

Networking resources provisioning

Article • 04/20/2023

For design considerations, see Networking cost estimates.

Azure Front Door

Outbound data transfers, inbound data transfers, and routing rules all affect Azure Front Door billing. Pricing information doesn't include the cost of accessing data from the backend services and transferring it to Azure Front Door. Those costs are billed based on data transfer charges described in Bandwidth pricing.

Another consideration is Web Application Firewall (WAF) settings. Adding policies drives up the cost.

For more information, see Azure Front Door pricing.

Azure Front Door reference architecture

Highly available multi-region web application uses Azure Front Door to route incoming requests to the primary region. If the application that runs the primary region becomes unavailable, Azure Front Door fails over to the secondary region.

Azure Application Gateway

There are two main pricing models:

- Fixed price

  You're charged from the time that you provision the gateway and it becomes available. You're charged for the amount of data that the gateway processes. For more information, see Application Gateway pricing.

- Consumption price

  This model applies to v2 SKUs that offer more features such as autoscaling, Azure Kubernetes Service Ingress Controller, and zone redundancy. You're charged based on the consumed capacity units. The capacity units measure the compute resources, persistent connections, and throughput. Consumption price is charged in addition to the fixed price.

For more information, see:

- Scaling Application Gateway v2 and WAF v2
- Application Gateway pricing

Application Gateway reference architecture

- Microservices architecture on Azure Kubernetes Service uses Application Gateway as the ingress controller.

- Securely managed web applications uses Application Gateway as a web traffic load balancer operating at Layer 7 that manages traffic to the web application. Web Application Firewall is enabled to enhance security.

Azure ExpressRoute

There are two main pricing models:

- Metered Data plan

  There are two pricing tiers: *Standard* and *Premium*, which is priced higher. The tier pricing is based on the circuit bandwidth.

  If you don't need to access the services globally, choose Standard. With this tier, you can connect to regions within the same zone at no extra cost. Outbound cross-zonal traffic incurs more cost.

- Unlimited Data plan

  All inbound and outbound data transfer is included in the flat rate. There are two pricing tiers: *Standard* and *Premium*, which is priced higher.

Calculate your usage and choose a billing plan. If you exceed about 68% of utilization, we recommend the *Unlimited Data plan*.

For more information, see Azure ExpressRoute pricing .

ExpressRoute reference architecture

Connect an on-premises network to Azure using ExpressRoute connects an Azure virtual network and an on-premises network connected using with VPN gateway failover.

Azure Firewall

Azure Firewall usage is charged at a fixed rate per deployment hour. There's extra cost for the amount of data transferred.

There aren't extra costs for a firewall deployed in an availability zone. There are extra costs for inbound and outbound data transfers associated with availability zones.

When compared to network virtual appliances (NVAs), with Azure Firewall you can save up to 30-50%. For more information, see Azure Firewall and network virtual appliances.

Azure Firewall reference architecture

- Hub-spoke network topology in Azure
- Deploy highly available NVAs

Azure Load Balancer

The Load Balancer service distributes inbound traffic according to the configured rules.

There are two tiers: *Basic* and *Standard*. The *Basic* tier is free.

For the *Standard* tier, you're charged only for the number of configured load-balancing and outbound rules. Inbound NAT rules are free. There's no hourly charge for the load balancer when no rules are configured.

For more information, see Azure Load Balancer pricing.

Load Balancer reference architecture

- Connect an on-premises network to Azure using ExpressRoute: Multiple subnets are connected through Azure load balancers.
- SAP S/4HANA in Linux on Azure: Distribute traffic to virtual machines in the application-tier subnet.
- Configure ExpressRoute and Site-to-Site coexisting connections: Network traffic from the VPN gateway is routed to the cloud application through an internal load balancer. The load balancer resides in the front-end subnet of the application.

Azure VPN Gateway

When provisioning a VPN Gateway resource, choose between two gateway types:

- VPN Gateway sends encrypted traffic across the public internet. Site-to-site, point-to-site, and virtual network-to-virtual network connections all use a VPN gateway.
- ExpressRoute gateway sends network traffic on a private connection. Azure ExpressRoute uses this configuration.

For VPN gateway, select *Route-based* or *Policy-based* depending on your VPN device and the kind of VPN connection you want to create. A route-based gateway allows point-to-site, inter-virtual network, or multiple site-to-site connections. Policy-based only allows one site-to-site tunnel. Point-to-site isn't supported. So, route-based VPN is more expensive.

You need to choose the SKU for Route-based VPN:

- For developer and test workloads, use *Basic*.
- For production workloads, an appropriate *Generation1* or *Generation2* SKU.

Each SKU has a range and pricing depends on the type of VPN gateway. Each type offers different levels of bandwidth, site-to-site, and point-to-site tunnel options. Some of those types also offer availability zones, which are more expensive. If you need higher bandwidth, consider Azure ExpressRoute.

VPN Gateway can be the cost driver in a workload because charges are based on the amount of time that the gateway is provisioned and available.

All inbound traffic is free. All outbound traffic is charged as per the bandwidth of the VPN type. Bandwidth also varies depending on the billing zone.

For more information, see:

- Hybrid connectivity
- VPN Gateway pricing
- Traffic across billing zones and regions
- Bandwidth pricing

VPN Gateway reference architecture

- Configure ExpressRoute and Site-to-Site coexisting connections connects the virtual network to the on-premises network through a VPN device.

Azure Traffic Manager

Azure Traffic Manager uses DNS to route and load balanced traffic to service endpoints in different Azure regions. An important use case is disaster recovery. In a workload, you can use Azure Traffic Manager to route incoming requests to the primary region. If that region becomes unavailable, Azure Traffic Manager fails over to the secondary region. There are other features that can make the application highly responsive and available. Those features cost money.

- Determine the best web app to handle request based on geographic location.
- Configure caching to reduce the response time.

Azure Traffic Manager isn't charged for bandwidth consumption. Billing is based on the number of DNS queries received, with a discount for services receiving more than 1 billion monthly queries. You're also charged for each monitored endpoint.

Azure Traffic Manager reference architecture

Multi-region N-tier application uses Azure Traffic Manager to route incoming requests to the primary region. If that region becomes unavailable, Azure Traffic Manager fails over to the secondary region. For more information, see the section Traffic Manager configuration.

DNS query charges

Azure Traffic Manager uses DNS to direct clients to specific services.

DNS queries that reach Azure Traffic Manager are charged in million query units.

Not all DNS queries reach Azure Traffic Manager. Recursive DNS servers run by enterprises and ISPs first attempt to resolve the query by using cached DNS responses. Those servers query Azure Traffic Manager at a regular interval to get updated DNS entries. That interval value or time to live (TTL) is configurable in seconds.

TTL can affect cost. A longer TTL value increases the amount of caching and reduces DNS query charges. Conversely, shorter a TTL value results in more queries.

Increased caching also affects how often the endpoint status refreshes. For example, the user failover times, for an endpoint failure, becomes longer.

Health monitoring charges

When Azure Traffic Manager receives a DNS request, it chooses an available endpoint based on configured state and health of the endpoint. To do this, Azure Traffic Manager continually monitors the health of each service endpoint.

The number of monitored endpoints are charged. You can add endpoints for services hosted in Azure and then add on endpoints for services hosted on-premises or with a different hosting provider. The external endpoints are more expensive, but health checks can provide high-availability applications that are resilient to endpoint failure, including Azure region failures.

Real User Measurement charges

Real User Measurements evaluates network latency from the client applications to Azure regions. That influences Azure Traffic Manager to select the best Azure region in which the application is hosted. The number of measurements sent to Azure Traffic Manager is billed.

Traffic View charges

By using Traffic View, you can get insight into the traffic patterns where you have endpoints. The charges are based on the number of data points used to create the insights presented.

Azure Virtual Network

Azure Virtual Network is free. You can create up to 50 virtual networks across all regions within a subscription. Here are a few considerations:

- Inbound and outbound data transfers are charged per the billing zone. Traffic that moves across regions and billing zones is more expensive. For more information, see:
  - Traffic across zones
  - Bandwidth pricing
- Virtual Network peering has an extra cost. Peering within the same region is cheaper than peering between regions or Global regions. Inbound and outbound traffic is charged at both ends of the peered networks. For more information, see Peering.
- Managed services don't always need a virtual network. The cost of networking is included in the service cost.

Web application cost estimates

Article • 04/18/2023

All web applications (apps) have no up-front cost or termination fees. Some charge only for what you use and others charge per-second or per-hour. In addition, all web apps run in Azure App Service plans. Together these costs can help determine the total cost of a web app.

Use the Azure pricing calculator to help create various cost scenarios.

App Service plans

App Service plans include the set of compute resources needed for the web app to run. Except for with the Free tier, an App Service plan carries a charge on the compute resources it uses. For a description of App Service plans, see Azure App Service plan overview.

You can potentially save money by choosing one or more App Service plans. To help find the solution that meets your business requirements, see Should I put an app in a new plan or an existing plan?

The *pricing tier* of an App Service plan determines what App Service features you get and how much you pay for the plan. The higher the tier, the higher the cost. For details on the pricing tiers, see App Service pricing.

You don't get charged for using the App Service features that are available to you, such as configuring custom domains, TLS/SSL certificates, deployment slots, and backups. For a list of exceptions, see How much does my App Service plan cost?

App Service cost

App Service plans are billed on a per second basis. If your solution includes several App Service apps, consider deploying them to separate App Service plans. This approach enables you to scale apps independently because they run on separate instances. This approach saves unnecessary cost.

Azure App Service supports two types of TLS/SSL connections: Server Name Indication (SNI) TLS/SSL Connections and IP Address TLS/SSL Connections. SNI-based TLS/SSL works on modern browsers while IP-based TLS/SSL works on all browsers. If your business requirements allow, use SNI-based TLS/SSL instead of IP-based TLS/SSL. There's no charge for SNI-based TLS/SSL. IP-based TLS/SSL incurs a cost per connection.

For more information, see SSL Connections in App Service pricing.

API Management cost

If you want to publish APIs hosted on Azure, on-premises, and in other clouds more securely, reliably, and at scale, use Azure API Management. The pricing tier you choose depends on the features needed based on your business requirements. For example, depending on your requirements for scalability, the number of units required to scale out range from 1 to 10. As the number of units increases, the cost of the pricing tier increases.

Self-hosted gateways are available in the Developer and Premium pricing tiers. There's an extra cost for this service if you use the Premium tier. There's no cost for self-hosted gateways if you use the Developer tier.

For pricing details, see API Management pricing.

Content Delivery Network cost

If you want to offer users optimal online content delivery, use Azure Content Delivery Network (CDN) in your workload. The data size that you use has the most significant effect on cost. Choose based on your business requirements. Purchasing data in increments of TB is significantly higher than purchasing data in increments of GB.

The provider you choose also can affect cost. Choose Microsoft as the provider to get the lowest cost.

Some data, such as shopping carts, search results, and other dynamic content, isn't cacheable. CDN offers Acceleration Data Transfers, also called Dynamic Site Acceleration (DSA), to accelerate this type of web content. The price for DSA is the same across Standard and Premium profiles.

For pricing details, see Content Delivery Network pricing.

Azure Cognitive Search cost

To set up and scale a search experience quickly and cost-effectively, use Azure Cognitive Search. When you create an Azure Cognitive Search service, a resource is created at a pricing tier or SKU that's fixed for the lifetime of the service. Billing depends on the type of service. For a list of services, see What are Azure Cognitive Services.

The charges are based on the number of transactions for each type of operation specific to a service. Some transactions are free. If you need more transactions, choose from the Standard instances.

For pricing details, see Azure Cognitive Search pricing.

Azure SignalR Service cost

Use SignalR for any scenario that requires pushing data from server to client in real time. For example, Azure SignalR Service provides secure and simplified communication between client and app one-to-one, such as a chat window. It also provides communication for one-to-many, such as instant broadcasting, IoT dashboards, or notification to social network. To learn more about Azure SignalR Service, see What is Azure SignalR Service?

We don't recommend the Free tier for a production environment. With the Standard tier, you pay only for what you use. We recommend the Standard tier as an enterprise solution because it offers a large number of concurrent connections and messages.

For pricing details, see Azure SignalR Service pricing.

Notification Hubs cost

To broadcast push notifications to millions of users at once or tailor notifications to individual users, use Azure Notification Hubs. Pricing is based on number of pushes. The Free tier is a good starting point for exploring push capabilities but we don't recommend it for production apps. If you require more pushes and features such as scheduled pushes or multi-tenancy, you can buy more pushes per million.

For more information, see Push notifications with Azure Notification Hubs. For pricing details, see Notification Hubs pricing.

Checklist - Monitor cost

Article • 04/20/2023

Use this checklist to monitor the cost of the workload.

- **Gather cost data from diverse sources to create reports**. Start with tools like Azure Advisor, Advisor Score, and Azure Cost Management. Build custom reports relevant for the business by using Consumption APIs.
  - Cost reports
  - Review costs in Cost analysis
- **Use resource tag policies to build reports**. Use tags to identify the owners of systems or applications and create custom reports.
  - Follow a consistent tagging standard
  - Video: How to review tag policies with Azure Cost Management
- **Use Azure built-in roles for cost**. Give access only to users who you want to view and analyze cost reports. You can define roles by their scope. For example, use the **Cost Management Reader role** to enable users to view costs for their resources in subscriptions or resource groups.
  - Provide the right level of cost access
  - Azure RBAC scopes
- **Respond to alerts and have a response plan according to the constraints**. Respond to alerts quickly and identify possible causes and any required action.
  - Budget and alerts
  - Use cost alerts to monitor usage and spending
- **Adopt both proactive and reactive approaches for cost reviews**. Conduct cost reviews at a regular cadence to determine the cost trend. Also, review reports that are created because of alerts.
  - Conduct cost reviews
  - Participate in central governance cost reviews
- **Analyze the cost at all scopes by using Cost analysis**. Identify services that drive the cost through different dimensions, such as location, usage meters, and so on. Review whether certain optimizations bring results. For example, analyze costs associated with reserved instances, savings plans, and Spot virtual machines (VMs) against business goals.
  - Quickstart: Explore and analyze costs with Cost analysis

- **Detect anomalies and identify changes in business or applications that might have contributed changes in cost**. Focus on these factors:
  - Traffic pattern as the application scales
  - Budget for the usage meters on resources
  - Performance bottle necks
  - CPU utilization and network throughput
  - Storage footprint for blobs, backups, and archiving
- **Use Visualization tools to analyze cost information**.
  - Create visuals and reports with in Power BI Desktop
  - Cost Management App

Set budgets and alerts

Article • 04/18/2023

Azure Cost Management has an alert feature. Alerts are generated when consumption reaches a threshold.

Consider the metrics for each resource in the workload. For each metric, build alerts on baseline thresholds. Admins are notified when the workload is using the services at capacity. The admins can then tune the resources to target SKUs based on current load.

You can also set alerts on allowed budgets at the resource group or management groups scopes. By setting alerts on metrics and budgets, you can balance both cloud services performance and budget requirements.

Over time, you can optimize the workload to autoheal itself when it triggers alerts. For more information about using alerts, see Use cost alerts to monitor usage and spending.

Respond to alerts

When you receive an alert, check the current consumption data because alerts don't trigger in real time. There might be a delay between the alert and the current actual cost. Look for significant difference between cost values when the alert occurred and the current cost. Next, conduct a cost review to discuss the cost trend, possible causes, and any required action.

Determine short and long-term actions that justify business value. Can a temporary increase in the alert threshold be a feasible fix? Do you need to increase the longer-term budget? Approval of any budget increase is mandatory.

If unnecessary or expensive resources trigger the alert, implement other Azure Policy controls. You can also add budget automation to trigger resource scaling or shutdowns.

Once you've created your alerts, your can also proactively review your Azure Costs and the forecast for upcoming weeks and months. For more information about Azure Cost Analysis and how to create custom views for your application, see Start using Cost analysis.

Revise budgets

After you identify and analyze your spending patterns, you can set budget limits for applications or business units. You want to assign access to view or manage each budget to the appropriate groups. Setting several alert thresholds for each budget can help track your burn down rate.

Azure Cost Management also has a tool that helps to identify cost anomalies and unexpected charges to your environment. To learn how to identify anomalies and unexpected changes in cost, see Identify anomalies and unexpected changes in cost

Generate cost reports

Article • 04/03/2023

To monitor the cost of the workload, use Azure cost tools or custom reports. You can scope reports to business units, applications, IT infrastructure shared services, and so on. Make sure that the information is consistently shared with the stakeholders.

Azure cost tools

Azure provides the following cost tools that help track cloud spending and offer recommendations.

- Azure Advisor
- Advisor Score
- Azure Cost Management
- Azure Cost Management Power BI app

Cost analysis

Cost analysis is a tool in Azure Cost Management that lets you view aggregated costs over a period of time. This view helps you understand your spending trends.

You can view costs at different scopes, such as for a resource group or specific resource tags. Cost analysis provides built-in charts and custom views. You can also download the cost data in CSV format to analyze with other tools.

For more information, see Quickstart: Explore and analyze costs with cost analysis.

> ⓘ **Note**
>
> There are many ways of purchasing Azure services. Not all services are supported by Azure cost management. For example, you obtain detailed billing information for services purchased through a Cloud Solution Provider (CSP) directly from the CSP. For more information about the supported cost data, see **Understand cost management data**.

Advisor recommendations

Azure Advisor recommendations for cost highlight any over-provisioned services and ways to lower costs. For example, it can show virtual machines that should be resized to a lower SKU, un-provisioned ExpressRoute circuits, and idle virtual network gateways.

For more information, see Advisor cost management recommendations.

Consumption APIs

Granular and custom reports help track cost over time. Azure provides a set of consumption APIs to generate such reports. These APIs let you query and create various cost data. Data includes usage data for Azure services and third-party services through Azure Marketplace, balances, budgets, recommendations on reserved instances, and others. You can configure Azure role-based access control (Azure RBAC) policies to let only a certain set of users or applications access the data.

For example, you want to determine the cost of all resources used in your workload for a given period. One way of getting this data is by querying usage meters and the rate of those meters. You also need to know the billing period of the usage. By combining these APIs, you can estimate the consumption cost.

The Azure consumption APIs include:

- Billing account API: To get your billing account to manage your invoices, payments, and track costs.
- Billing periods API: To get billing periods that have consumption data.
- Usage detail API: To get the breakdown of consumed quantities and estimated charges.
- Marketplace store charge API: To get usage-based marketplace charges for third-party services.
- Price sheet API: To get the applicable rate for each meter.

You can import the results of the APIs into analysis tools.

> **Note**
>
> Consumption APIs are supported for enterprise enrollments and web direct subscriptions (with exceptions). Check **Consumption APIs** for updates to determine support for other types of Azure subscriptions.

For more information about common cost scenarios, see Billing automation scenarios.

Custom scripts

Use Azure APIs to schedule custom scripts that identify orphaned or empty resources. For example, unattached managed disks, load balancers, application gateways, or Azure SQL Servers with no databases. These resources incur a flat monthly charge while unused. Other resources might be stale, for example, VM diagnostics data in blob or table storage. To determine if you should delete the item, check its last use and modification timestamps.

Analyze and visualize

Start with the usage details in the invoice. Review that information against relevant business data and events. If there are anomalies, evaluate the significant changes in business or applications that might have contributed those changes.

Power BI Desktop has a connector that lets you connect billing data from Azure Cost Management. You can create custom dashboards and reports, ask questions of your data, and publish and share your work.

> ⓘ Note
>
> Sharing requires Power BI Premium licenses.

For more information, see Create visuals and reports with the Azure Cost Management connector in Power BI Desktop.

You can also use the Cost Management App. The app uses the Azure Cost Management Template app for Power BI. You can import and analyze usage data and incurred cost within Power BI.

Azure Cost Management Power BI app

Use the Azure Cost Management Power BI app to analyze and manage your costs in Power BI. You can use the app to monitor costs and usage trends, produce reports, and identify cost optimization options to reduce expenditures.

The Cost Management Power BI app currently supports customers with an Enterprise Agreement. For more information, see Azure Cost Management Power BI app.

Conduct cost reviews

Article • 04/18/2023

Cost monitoring tracks and reviews cloud costs to help establish budget controls and prevent misuse. It's important to adopt proactive and reactive review approaches for monitoring cost. Stakeholders should conduct cost reviews regularly and include reactive cost reviews. For example, when a budget limit causes an alert.

Who should be included in a cost review?

Cost monitoring is a complex process that involves planning, cost estimation, budgeting, and cost control, which can include several different teams.

- **Cloud Architect/Administrator**. Monitor and back-up cloud systems.
- **Product Owner**. Needs awareness of the decisions.
- **Finance Owner and Financial Operations Practitioner**. Need to understand cloud billing to derive business benefits using financial metrics to make effective decisions.

You can identify owners of systems or applications through resource tags.

Plan and schedule cost reviews

We recommend scheduling cost reviews as part of the regular business reviews.

- Schedule a review during the billing period. This review creates awareness of the estimated pending billing. You can use Azure Advisor, Advisor Score, and Azure Cost Management – cost analysis to help with the reports.
- Schedule a review after the billing period. This review shows the actual cost with activity for that month. Use TimeframeType to generate monthly reports. The APIs can query data that gets information on balances, new purchases, Azure Marketplace service charges, adjustments, and overage charges. For more information, see Cost Management automation overview.
- Schedule a review when you receive a budget alert or Azure Advisor recommendation.

Web Direct (pay-as-you-go) and Cloud Solution Provider (CSP) billing occurs monthly, while Enterprise Agreement (EA) billing occurs annually. But ensure you do a cost review regardless of the billing cycle.

Checklist - Optimize cost

Article • 04/12/2023

Continue to monitor and optimize the workload by using the right resources and sizes. Use this checklist to optimize a workload.

- **Review the underutilized resources.** Evaluate CPU utilization and network throughput over time to check if the resources are used adequately. Azure Advisor identifies underutilized virtual machines (VMs). You can choose to decommission, resize, or shut down the machine to meet the cost requirements.
  - Resize virtual machines
  - Shutdown the underutilized instances

Auto start and stop VMs tool to non-production VMs. The start and stop VMs v2 feature starts or stops Azure Virtual Machines instances across multiple subscriptions. It starts or stops virtual machines on user-defined schedules. For more information, see Start and stop VMs.

- **Continuously take action on the cost reviews**. Treat cost optimization as a process, rather than a point-in-time activity. Use tooling in Azure that provides recommendations on usage or cost optimization. Review the cost management recommendations and take action. Make sure that all stakeholders are in agreement about the implementation and timing of the change.
  - **Recommendations for cost management** in the Azure portal
  - Recommendations in the Cost Management Power BI app
  - Recommendations in Azure Advisor
  - Recommendations using Reservation REST APIs

- **Use savings plans**. By using savings plans, you get the most flexible savings for dynamic workloads, while accommodating planned or unplanned changes. With savings plans, you commit to a fixed, hourly, dollar amount, collectively, on compute services, globally.
  - Savings plans

- **Use Azure reservations**. You get the greatest cost savings for stable, predictable workloads with no planned changes by using reserved instances or capacity. Consider that you're often committing to specific SKUs, throughput, or utilization targets, often in a particular Azure region.
  - Azure reservations

- **Use discount prices**. The following methods of buying Azure resources can lower costs.

- Azure Hybrid Benefit
- Azure reservations

There are also payment plans offered at a lower cost:

- Microsoft Azure Enterprise portal
- Enterprise Dev Test Subscription
- Microsoft Cloud Solution Provider program

- **Have a scale-in and scale-out policy**. In a cost-optimized architecture, costs scale linearly with demand. Increasing customer base shouldn't require more investment in infrastructure. Conversely, if demand drops, scale-down unused resources. And autoscale Azure resources when possible.
  - Autoscale instances
  - Azure SQL Database elastic pools are a simple, cost-effective solution for managing and scaling multiple databases that have varying and unpredictable usage demands
  - SQL Serverless is a compute tier for single databases in Azure SQL Database that automatically scales compute based on workload demand and bills for the amount of compute that you use per second.

- **Reevaluate design choices**. Analyze the cost reports and forecast the capacity needs. You might need to change some design choices.

  - **Choose the right storage tier**. Consider using hot, cool, and archive tiers for storage account data. Storage accounts can provide automated tiering and lifecycle management. For more information, see Review your storage options.

  - **Choose the right data store**. Instead of using one data store service, use a mix of data store depending on the type of data you need to store for each workload. For more information, see Choose the right data store.

  - **Choose Spot VMs for low priority workloads**. Spot VMs are ideal for workloads that can be interrupted, such as highly parallel batch processing jobs.
    - Spot VMs

  - **Optimize data transfer**. Only deploy to multiple regions if your service levels require it for either availability or geo-distribution. Data going out of Azure data centers can add cost because pricing is based on billing zones.
    - Traffic across billing zones and regions

  - **Reduce load on servers**. Use Azure Content Delivery Network (CDN) and caching service to reduce load on front-end servers. Caching is suitable for servers that are continually rendering dynamic content that doesn't change frequently.

- **Use managed services**. Measure the cost of maintaining infrastructure and replace it with Azure platform as a service (PaaS) or software as a service (SaaS) services.
  - Managed services

Autoscale instances

Article • 04/20/2023

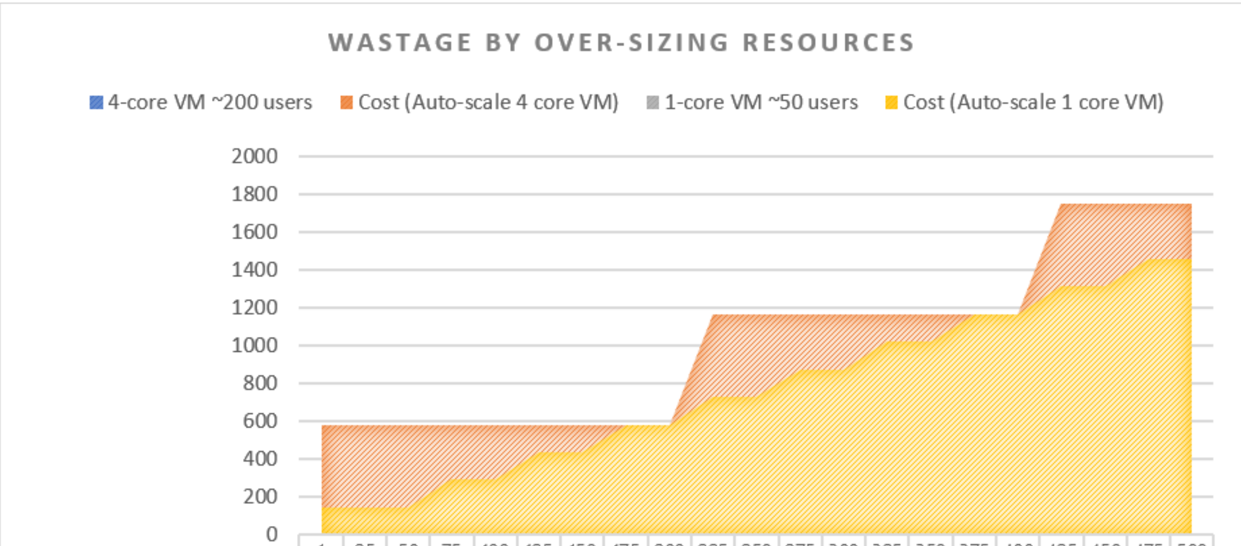
In Azure, it's easier to grow a service with little to no downtime compared to downscaling a service, which usually requires deprovisioning or downtime. In general, opt for scale-out instead of scale up.

For certain applications, capacity requirements might change over time. Autoscaling policies allow for less error-prone operations and cost savings through robust automation.

Virtual machine instances

For autoscaling, consider the choice of instance size. The size can significantly change the cost of your workload.

> Choose smaller instances where workload is highly variable and scale out to get the desired level of performance, rather than up. This approach lets you make your cost calculations and estimates granular.



	1	25	50	75	100	125	150	175	200	225	250	275	300	325	350	375	400	425	450	475	500
4-core VM ~200 users	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	3	3	3	3
Cost (Auto-scale 4 core VM)	584	584	584	584	584	584	584	584	584	1168	1168	1168	1168	1168	1168	1168	1168	1752	1752	1752	1752
1-core VM ~50 users	1	1	1	2	2	3	3	4	4	5	5	6	6	7	7	8	8	9	9	10	10
Cost (Auto-scale 1 core VM)	146	146	146	292	292	438	438	584	584	730	730	876	876	1022	1022	1168	1168	1314	1314	1460	1460

Stateless applications

You can use many Azure services to improve the application's ability to scale dynamically, even if the application isn't originally designed to scale dynamically.

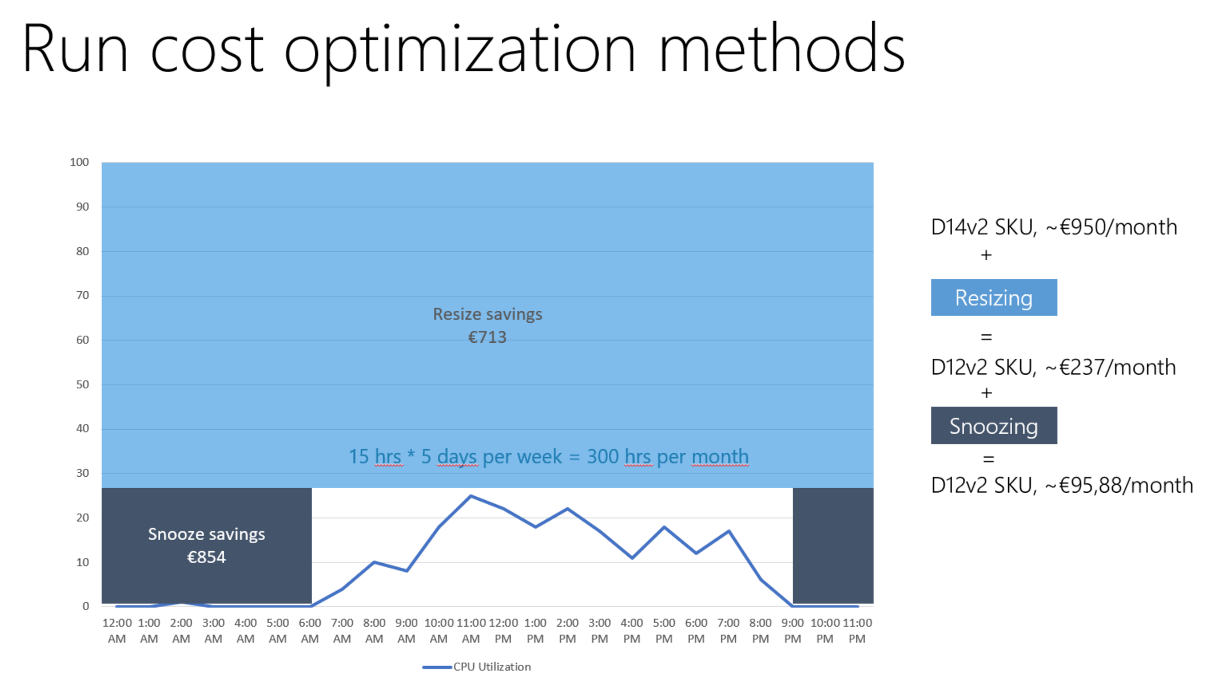
For example, many ASP.NET stateful web applications can be made stateless. Then you can autoscale them, which results in a cost benefit. You store state using Azure Redis Cache, or Azure Cosmos DB as a back-end session state store through a session state provider.

Virtual machines

Article • 11/30/2022

Virtual machines can be deployed in fix-sized blocks. These VMs must be adequately sized to meet capacity demand and reduce waste.

For example, look at a VM running the SAP on Azure project can show you how initially the VM was sized based on the size of existing hardware server (with cost around €1 K per month), but the real utilization of VM wasn't more than 25% - but simple choosing the right VM size in the cloud we can achieve 75% saving (resize saving). And by applying the snoozing you can get additional 14% of economy:



It's easy to handle cost comparison when you're well equipped and for this Microsoft provides the set of specific services and tools that help you to understand and plan costs. These include the TCO Calculator, Azure Pricing Calculator, Azure Cost Management (Cloudyn), Azure Migrate, Azure Cosmos DB Sizing Calculator, and the Azure Site Recovery Deployment Planner.

Here are some strategies that you can use to lower cost for virtual machines.

Resize virtual machines

You can lower cost by managing the size and the number of VMs.

> Determine the load by analyzing the CPU utilization to make sure that the instance is adequately utilized.

Ideally, with the right size, the current load should fit in a lower SKU of the same tier. Another way is to lower the number instances and still keep the load below a reasonable utilization. Azure Advisor recommends load less than 80% utilization for non-user facing workloads and 40% when user-facing workload. It also provides current and target SKU information.

You can identify underutilized machines by adjusting the CPU utilization rule on each subscription.

Resizing a virtual machine does require the machine to be shut down and restarted. There might be a period of time when it will be unavailable. Make sure you carefully time this action for minimal business impact.

Shut down underutilized instances

Use the **Start/stop VMs during off-hours** feature of virtual machines to minimize waste. There are many configuration options to schedule the start and stop times. The feature is suitable as a low-cost automation option. For information, see Start/stop VMs during off-hours solution in Azure Automation.

Azure Advisor evaluates virtual machines based on CPU and network utilization over a time period. Then, the recommended actions are shut down or resize instances and cost saving with both actions.

Spot VMs

Some workloads don't have a business requirement to complete a job within a period.

Can the workload be interrupted?

Spot VMs are ideal for workloads that can be interrupted, such as highly parallel batch processing jobs. These VMs take advantage of the surplus capacity in Azure at a lower cost. They're also well suited for experimental, development, and testing of large-scale solutions.

For more information, see Use Spot VMs in Azure or check out our Azure Virtual Machine Spot Eviction guide to learn how to create a reliable interruptible workload in Azure.

Reserved VMs

Virtual machines are eligible for Azure Reservations. You can prepay for VM instances if you can commit to one or three years. Reserved instances are appropriate for workloads that have a long-term usage pattern.

The discount only applies to compute and not the other meters used to measure usage for VMs. The discount can be extended to other services that emit VM usage, such as Virtual machine scale sets and Container services, to name a few. For more information, see Software costs not included with Azure Reserved VM Instances and Services that get VM reservation discounts.

With reserved instances, you need to determine the VM size to buy. Analyze usage data using **Reservations Consumption APIs** and follow the recommendations of Azure portal and Azure Advisor to determine the size.

Reservations also apply to dedicated hosts. The discount is applied to all running hosts that match the reservation scope and attributes. An important consideration is the SKU for the host. When selecting a SKU, choose the VM series and type eligible to be a dedicated host. For more information, see Azure Dedicated Hosts pricing .

For information about discounts on virtual machines, see How the Azure reservation discount is applied to virtual machines.

Savings plans

Virtual machines are eligible for savings plans. By committing to a one-year or three-year savings plan, you spend a fixed hourly dollar amount collectively on all compute services. Consider that the commitment spans across all participating compute services globally.

Using savings plans can save you up to 66% on cost.

Caching data

Article • 04/20/2023

Caching is a process that stores a copy of the data in front of the main data store. The cache store locations are closer to the consumer than the main store. Advantages of caching include faster response times and the ability to serve data quickly, which saves on the overall cost. Check the built-in caching features of Azure services used in your architecture. Azure also offers caching services such as Azure Cache for Redis or Azure CDN.

For information about what data is suitable for caching, see Caching guidance.

Lower costs associated with reliability and latency

Caching is a cost-effective way to store data, provide reliability, and reduce network latency.

- The type of data helps you determine if you need the complex capabilities of the backend data store, such as data consistency. For fully static data, store it in a caching store. If the data doesn't change frequently, consider placing a copy of the data in a caching store and refresh it from time to time.

  > For example, an application stores images in blob storage. Then every time the application requests an image, the business logic generates a thumbnail from the main image and returns it to the caller. If the main image doesn't change too often, then return the previously generated thumbnails stored in a cache. This way, you can save on resources required to process the image and lower the request response rate.

- If the backend is unavailable, the cache continues to manage requests by using the copy until the backend fails over to the backup data store.

- Caching is also an effective way of reducing network latency. Instead of reaching the central server, the response uses cache. That way, the client can receive the response instantaneously. If you need higher network performance and the ability to support more client connections, choose a higher tier of the caching service. However, higher tiers incur more costs.

Caching can be expensive

Incorrect use of caching can result in severe business outcomes and higher costs.

- If you choose to add a cache, your architecture will have multiple data stores. There are added costs to keeping them coordinated. You might need to fill the cache before putting it in production. If you fill the cache on the application's first access, it can introduce latency. If you seed the cache, it can affect the application's start time. If you don't refresh the cache, your customers can get stale data.

  Invalidate the cache at the right time when there's latest information in the source system. Use strategies to age out the cache when appropriate.

- Add instrumentation to make sure the caching layer is working optimally. Instrumentation adds complexity and implementation cost.

Caching services such as Azure Cache for Redis offer tiers by cost. Pricing per tier is based on the cache size and network performance. A smaller cache increases latency. Before you choose a tier, estimate a baseline. Try load testing the number of users and cache size.

Tradeoffs for cost

Article • 04/25/2023

As you design a workload, consider tradeoffs between cost optimization and other aspects of the design, such as security, scalability, resilience, and operability.

What is most important for the business: lowest cost, no downtime, high throughput?

An optimal design doesn't equate to a low-cost design. You might make risky choices in favor of a cheaper solution.

Cost versus reliability

Cost has a direct correlation with reliability.

Does the cost of high availability components exceed the acceptable downtime?

Overall service level agreement (SLA), Recovery Time Objective (RTO), and Recovery Point Objective (RPO) might lead to expensive design choices. If your service SLAs, RTOs, and RPOs times are short, greater investment is inevitable for high availability and disaster recovery options.

For example, to support high availability, you choose to host the application across regions. This choice is costlier than single region because of the replication costs or the need for provisioning extra nodes. Data transfer between regions also adds cost.

If the cost of high availability exceeds the cost of downtime, you can save by using Azure platform-managed replication and then recover data from the backup storage.

For resiliency, availability, and reliability considerations, see the Reliability pillar.

Cost versus performance efficiency

Boosting performance leads to higher costs.

Many factors affect performance.

- **Fixed or consumption-based provisioning**. Avoid cost estimating a workload at consistently high utilization. Consumption-based pricing is more expensive than the equivalent provisioned pricing. Smooth out the peaks to get a consistent flow of compute and data. Ideally, use manual scaling and autoscaling to find the right balance. Scaling up is more expensive than scaling out.

- **Azure regions**. Cost scales directly with the number of regions. Locating resources in cheaper regions shouldn't negate the cost of network ingress and egress or by degraded application performance because of increased latency.

- **Caching**. Every payload's render cycle consumes both compute and memory. Use caching to reduce load on servers and save with precanned storage and bandwidth costs. The savings can be dramatic, especially for static content services.
    - While caching can reduce cost, there are some performance tradeoffs. For example, Azure Traffic Manager pricing is based on the number of DNS queries that reach the service. Reduce that number through caching and configure how often the cache refreshes. Relying on the cache that isn't frequently updated causes longer user failover times if an endpoint is unavailable.

- **Batch or real-time processing**. Using dedicated resources for batch processing of long-running jobs increases the cost. You can lower cost by provisioning Spot VMs but be prepared for the job to be interrupted every time Azure evicts the VM.

For performance considerations, see the Performance efficiency pillar.

Cost versus security

Increasing security of the workload increases costs.

As a rule, don't compromise on security. With certain workloads, you can't avoid security costs. For example, for specific security and compliance requirements, deploying to differentiated regions is more expensive. Premium security features can also increase the cost. There are areas where you can reduce costs by using native security features. For example, avoid implementing custom roles if you can use built-in roles.

For security considerations, see the Security pillar.

Cost versus operational excellence

Investing in systems monitoring and automation might increase the cost initially but over time reduces cost. Integrate IT operations processes like user or application access provisioning, incident response, and disaster recovery with the workload.

Cost of maintaining infrastructure is more expensive. With platform as a service (PaaS) or software as a service (SaaS) services, infrastructure, platform management services, and other operational efficiencies are included in the service pricing.

For operational considerations, see the Operational excellence pillar.

Operational excellence documentation

Apply reliable, predictable, and automated operations processes to your architecture to keep an application running in production.

Key points

QUICKSTART

Principles

Automation overview

Application design

Monitoring checklist

DevOps culture

DevOps patterns

TRAINING

Operational Excellence

VIDEO

Getting Started with DevOps

Automate business solutions

GET STARTED

Activate resources on demand

Create repeatable and consistent environments

Configure programmatic infrastructure

Automate operational tasks

CONCEPT

Understand automation best practices

Deploy infrastructure with code

Bootstrap automation

Understand Azure Functions

TRAINING

Choose the best Azure service to automate your business processes

VIDEO

Azure DevOps with ARM templates

Engineer modern release management strategies

OVERVIEW

Configure development environments

Practice continuous integration

Perform release testing

Consider performance

Evaluate release deployment

Anticipate deployment issues

CONCEPT

Understand source control

Consider continuous integration pipelines

Automate testing

Optimize your build

Document your release processes

Stage your workloads

Roll back with deployment slots

VIDEO

Continuous Delivery and Release Management | DevOps for Mobile

Monitor for DevOps

CONCEPT

Understand application and platform monitoring

Analyze alerts

VIDEO

Continuous Monitoring for Web Perf and Accessibility

Monitoring AKS with Azure

DevOps tools and services

REFERENCE

Azure DevOps

Azure Diagnostics

Azure Monitor

Azure Resource Manager

Azure Role-Based Access Control (RBAC)

Bridge to Kubernetes

Chef

Docker Desktop

Git

Terraform

DevOps APIs

REFERENCE

Alerts - Get All

Automation REST API

Azure Container Instances

Azure Deployment Manager

Azure Resource Health

Network Monitoring

Overview of the operational excellence pillar

Article • 05/04/2023

The operational excellence pillar covers the processes that keep an application running in production. Deployments must be reliable and predictable. Automated deployments reduce the chance of human error. Fast and routine deployment processes don't slow down the release of new features or bug fixes. It's equally important to be able to quickly roll backward or forward if an update has problems.

To assess your workload by using the tenets found in the Microsoft Azure Well-Architected Framework, see the Azure Well-Architected Review.

We recommend the following video to help you achieve operational excellence with the Azure Well-Architected Framework:

https://learn.microsoft.com/shows/azure-enablement/achieve-operational-excellence-with-azure-well-architected-framework/player

Articles

The Microsoft Azure Well-Architected Framework includes the following articles in the operational excellence pillar:

Operational excellence articles	Description
Release engineering: application development	Provides guidance on how to design, build, and orchestrate workloads with operational excellence principles in mind.
Monitoring operations of cloud applications	Shows how monitoring and diagnostics are essential to any workload and are crucial for cloud applications that run in a remote datacenter.
Performance considerations for your deployment infrastructure	Describes the monitoring and management of performance and availability of software applications through operational excellence.
Release engineering: deployment	Shows how deploying your application code is one of the key factors that determines your application stability.

Operational excellence articles	Description
Repeatable Infrastructure	Refers to best practices for deploying the platform where your application runs. Infrastructure provisioning is frequently known as *Deployment Automation* or *Infrastructure as code*.
Testing your application and Azure environment	Shows how testing is fundamental for preparing for the unexpected and catching mistakes before they affect users.

Enforcing resource-level rules through Azure Policy helps ensure adoption of operational excellence best practices for all the assets, which supports your workload. For example, Azure Policy can help ensure all the virtual machines (VMs) supporting your workload adhere to a preapproved list of VM SKUs.

Azure Advisor provides a set of Azure Policy recommendations to help you quickly identify opportunities to implement Azure Policy best practices for your workload.

Use the DevOps checklist as a starting point to assess your DevOps culture and process.

Next steps

Reference the operational excellence principles to guide you in your overall strategy.

Principles

Operational excellence design principles

Article • 03/15/2023

The principles of operational excellence are a series of considerations that can help achieve superior operational practices.

To achieve a higher competency in operations, consider and improve how software is:

- Developed
- Deployed
- Operated
- Maintained

Equally important, provide a team culture, which includes:

- Experimentation and growth
- Solutions for rationalizing the current state of operations
- Incident response plans

To assess your workload using the tenets found in the Azure Well-Architected Framework, reference the Microsoft Azure Well-Architected Review.

The following design principles provide:

- Context for questions
- Why a certain aspect is important
- How an aspect is applicable to Operational excellence

These critical design principles are used as lenses to assess the Operational excellence of an application deployed on Azure. These lenses provide a framework for the application assessment questions.

Optimize build and release processes

Embrace software engineering disciplines across your entire environment, which include the following disciplines:

- Provision with Infrastructure as Code
- Build and release with continuous integration and continuous delivery (CI/CD) pipelines
- Use automated testing methods
- Avoid configuration drift through configuration as code

This approach ensures the creation and management of environments throughout the software development lifecycle. It enables:

- Consistency
- Repetition
- Early detection of issues

Understand operational health

Implement systems and processes to monitor all aspects of your workload. Including:

- Build and release processes
- Infrastructure health
- Application health

Robust monitoring ensures the observability of a workload and allows you to correlate events and take proactive mitigating issues.

In addition, customer data is critical to understanding the health of a workload and whether the service is meeting the business goals.

> ⓘ **Important**
>
> The Health Modeling section of the **Azure Mission-Critical** framework contains further in-depth guidance and examples on how to build a health model for a given workload.

Rehearse recovery and practice failure

Rehearse recovery and practice failure using the following methods:

- Run disaster recovery (DR) drills on a regular cadence.
- Use chaos engineering practices to identify and remediate weak points in application reliability.
- Rehearse failure to validate the effectiveness of recovery processes and ensure teams are familiar with their responsibilities.
- Document past failures and automate their remediation where possible.

Embrace continuous operational improvement

Teams that embrace continuous operational improvement continuously evaluate and refine operational procedures and tasks. They strive to reduce complexity and ambiguity whenever possible.

Adopting a continuous improvement culture helps organizations:

- Evolve processes over time.
- Optimize inefficiencies and associated processes.
- Learn from failures.
- Continuously evaluate new opportunities.

Use loosely coupled architecture

Use modern architecture patterns such as:

- microservices
- loosely coupled
- serverless

and pair this with cloud design patterns such as:

- Circuit breakers
- Load-Leveling
- Throttling

and advanced deployment strategies like:

- Canary
- Blue-green
- Staggered

to enable teams to build and deploy services independently and minimize the impact if there is a service failure.

This principle also extends to procedural decoupling. Teams will be able to take full advantage of their loosely coupled architecture if they do not have to depend on partner teams to support, approve, or operate their workloads.

Next step

Automation overview

Automation overview: Goals, best practices, and types

Article • 05/30/2023

Automation has revolutionized how businesses operate and this trend continues to evolve. Businesses have moved to automating manual processes so that engineers can focus attention on tasks that add business value. Automating business solutions lets you:

- Activate resources on demand.
- Deploy solutions rapidly.
- Minimize human error in configuring repetitive tasks.
- Produce consistent and repeatable results.

For more information, see Deployment considerations for automation.

Goals of automation

When you automate technical processes, a common approach for some organizations is to automate what they can and leave the more difficult processes for humans to perform manually.

A goal of automation is to make tools that do what humans can do, only better. For example, a human can perform any given task once. But when the task requires repetitive runs, especially over long time periods, an automated system is better equipped to do that work with more predictable, error-free results. Increasing speed is another goal in automation. When you practice these automation goals, you can build systems that are faster, repetitive, and can run on a daily basis.

Automate toil to improve efficiency

Most automation involves a percentage of *toil*. Toil is the operational work that's related to a process that's manual, repetitive, can be automated, and has minimal value. It's counterproductive to automation but in many organizations, a small amount of toil is unavoidable. It becomes an issue when too much toil slows progress. A project's production velocity decreases if engineers are continuously interrupted by manual tasks attributed to toil, either planned or unplanned. Too much toil can impact job satisfaction. Engineers become dissatisfied when they find themselves spending too much time on operational toil rather than on other projects.

Automation should be developed, and increased, so engineers can eliminate future toil. By reducing toil, engineers can concentrate on innovating business solutions.

For more information, see Toil automation.

Automation best practices

- **Ensure consistency:** The more manual processes are involved, the more prone to human error. Manual processes can lead to mistakes, oversights, reduction of data quality, and reliability problems.
- **Centralize mistakes:** Choose a platform that lets you fix bugs in one place in order to fix them everywhere. This best practice reduces the chance of error and the possibility of the bug being reintroduced.
- **Identify issues quickly:** Complex issue might not always be identifiable in a timely manner. But with good automation, detection of these issues should occur quickly.
- **Maximize employee productivity:** Automation leads to more innovative solutions, and in general provides more value to the business. This improvement in turn raises morale and job satisfaction. Once a process is automated, training and maintenance can be greatly reduced or eliminated. This shift frees engineers to spend less time on manual processes and more time on automating business solutions.

Types of automation

Three types of automation are described in this article:

- Infrastructure deployment
- Infrastructure configuration
- Operational tasks

These categories share the same goals and best practices mentioned previously. They differ in areas where Azure provides solutions that help achieve optimal automation. Other types of automation, such as continuous deployment and continuous delivery, are described further in the Operational Excellence pillar.

Infrastructure deployment

As businesses move to the cloud, they need to *repeatedly* deploy their solutions and know that their infrastructure is in a *reliable* state. To meet these challenges, you can automate deployments by using a practice referred to as infrastructure as code. In code, you define the infrastructure that needs to be deployed.

There are many deployment technologies you can use with Azure. Here are three examples:

- Azure Resource Manager (ARM) templates
- Azure Bicep
- Terraform

These technologies use a *declarative* approach. This approach lets you state what you intend to deploy without having to write the sequence of programming commands to create it. You can deploy not only virtual machines, but also the network infrastructure, storage systems, and any other resources you might need.

Infrastructure configuration

If you don't manage configuration carefully, your business could encounter disruptions such as systems outages and security issues. Optimal configuration enables you to quickly detect and correct configurations that could interrupt or slow performance.

When creating new resources on Azure, you can take advantage of configuration as code to bootstrap the deployment.

Configuration tools can also be used to configure and manage the ongoing state of deployed resources.

Operational tasks

As the demand for speed in performing operational tasks increases over time, you're expected to deliver things faster and faster. Manually performing operational tasks will fail to scale as demand increases. This is where automation can help. To meet on-demand delivery using an automation platform, you need to develop automation components (such as runbooks and configurations), create integrations to systems that are already in place efficiently, and operate and troubleshoot.

Advantages of automating operational tasks include:

- Optimize and extend existing processes.
- Deliver flexible and reliable services.
- Lower costs.
- Improve predictability.

Two popular options for automating operational tasks are:

- Azure Functions - Run code without managing the underlying infrastructure on where the code is run.
- Azure Automation - Uses programming and scripting language to automate operational tasks in code and run on demand.

For more information, see Automation.

Next steps

Automate repeatable infrastructure

Repeatable infrastructure

Article • 05/03/2023

Historically, deploying a new service or application involves manual work such as procuring and preparing hardware, configuring operating environments, and enabling monitoring solutions. Ideally, an organization would have multiple environments in which to test deployments. These test environments should be similar enough to production that deployment and run time issues are detected before deployment to production. This manual work takes time, is error-prone, and can produce inconsistencies between the environments if not done well.

Cloud computing changes the way we procure infrastructure. No longer are we unboxing, racking, and cabling physical infrastructure. We have internet accessible management portals and REST interfaces to help us. We can now provision virtual machines, databases, and other cloud services on demand and globally. When we no longer need cloud services, they can be easily deleted.

However, cloud computing alone doesn't remove the effort and risk in provisioning infrastructure. When you use a cloud portal to build systems, many of the same manual configuration tasks remain. Application servers require configuration, databases need networking, and firewalls need firewalling.

Azure landing zones (repeatable environment configuration)

Organizations that manage, govern, or support multiple workloads in the cloud will require repeatable **and consistent** environments. Azure landing zones provide central operations teams (also known as platform teams) with a repeatable approach to environmental configuration. To deliver consistent environments, all Azure landing zones provide a set of common design areas, reference architecture, reference implementation, and a process to modify that deployment to fit the organization design requirements.

This environment is primarily enabled by using the Azure landing zone design principles of policy-driven governance alongside subscription democratization.

The following links are from the Cloud Adoption Framework to help you deploy Azure landing zones:

- Azure landing zones adhere to a common set of design areas to guide configuration of required environment considerations including: Identity, Network

topology and connectivity, Resource organization, Governance disciplines, Operations baseline, and Business continuity and disaster recovery (BCDR)
- Azure landing zones can be deployed through via:
  - Azure landing zones accelerator portal
  - Azure landing zones - Bicep modules design considerations
  - Azure landing zones - Terraform modules design considerations

To get started with Azure landing zones to create consistent, repeatable environment configuration, see the article series on Azure landing zones.

Deploy infrastructure with code

To fully realize deployment optimization, reduce configuration effort, and automate full environments' deployment, something more is required. One option is referred to as infrastructure as code.

Infrastructure as code (IaC) is the management of infrastructure—such as virtual machines, load balancers, and connection topology—in a descriptive model, using a versioning system that's similar to what's used for source code. When you're creating an application, the same source code generates the same binary every time it's compiled. In a similar manner, an IaC model generates the same environment every time it's applied. IaC is a key DevOps practice, and it's often used with continuous delivery.

Ultimately, IaC lets you and your team develop and release changes faster, but with higher confidence in your deployments.

Gain higher confidence

One of the biggest benefits of IaC is the level of confidence you can have in your deployments, and in your understanding of the infrastructure and its configuration.

Integrate with your process. If you have a process by which code changes are peer reviewed, you can use that same process for reviewing infrastructure changes. This review process can help proactively detect problematic configurations that are difficult to see when making manual infrastructure changes.

Consistency. Following an IaC process ensures that the whole team follows a standard, well-established process. Historically, some organizations designate a single or small set of individuals responsible for deploying and configuring infrastructure. By following a fully automated process, responsibility for infrastructure deployments moves from individuals into the automation process and tooling. This move broadens the number of team members who can initiate infrastructure deployments while maintaining consistency and quality.

Automated scanning. Many types of IaC configurations can be scanned by automated tooling. One such type of tooling is linting to check for errors in the code. Another type will scan the proposed changes to your Azure infrastructure to ensure they follow security and performance best practices. Automated scanning can be an important part of a Continuous security approach.

Secret management. Most solutions require secrets to be maintained and managed. These include connection strings, API keys, client secrets, and certificates. Following an IaC approach means that you need to adopt best practices for managing secrets. For example, Azure Key Vault is used to store secrets securely. It integrates with many IaC tools and configurations to ensure that the person conducting the deployment doesn't need access to production secrets. This approach, in turn, helps you adhere to the security principle of least privilege.

Access control. A fully automated IaC deployment pipeline means that an automated process should perform all changes to your infrastructure resources. This approach has many security benefits. For more information, see Security design principles. By automating your deployments, you can be confident that changes deployed to your environment have followed the correct procedure. You can even consider expanding the number of people who can initiate a deployment since the deployment itself is done in a fully automated way. Ideally, you would remove the ability for humans to manually modify your cloud resources and instead rely completely on automated processes. In emergencies, you can allow for this ability to be overridden, by using a break glass account or Privileged identity management.

Avoid configuration drift. When you use IaC, you can redeploy all of your environment templates on every release of your solution. IaC tooling is built to be idempotent, which means that it can run several times and will produce the same result each time.

Running infrastructure as code operations frequently has the following benefits:

- Avoids deployment staleness, which prevents unforeseen issues during a redeployment. For example, as part of a disaster recovery plan.
- Reduces complexity overall as there's one process that's rehearsed often.
- Helps to avoid configuration drift. Accidental changes outside of the regular pipeline are corrected quickly and the source of truth for your environment's configuration remains in code.

Manage multiple environments

Many organizations maintain multiple environments, for example, test, staging, and production. In some cases, multiple production environments are maintained for multi-tenanted solutions or geographically distributed applications. Ensuring consistency across these environments can be difficult; using infrastructure as code solutions can help.

Manage non-production environments. A common pain point for organizations is when nonproduction environments are dissimilar to production environments. Often, when you build production and nonproduction environments by hand, the configuration of each won't match. This mismatch slows down the testing of changes and reduces confidence that changes won't harm a production system. When following an IaC approach, this problem is minimized. When you use IaC automation, the same infrastructure configuration files can be used for all environments, producing almost identical environments. When needed, differentiation can be achieved by using input parameters for each environment.

Dynamically provision environments. Once you have your IaC configurations defined, you can use them to provision new environments more efficiently. This agility can be enormously helpful when you're testing your solution. For example, you could quickly provision a duplicate of your production environment that can then be used for security penetration tests, load testing or help a developer track down a bug.

Scale production environments. IaC configurations can be used to deploy more instances of your solution, ensuring consistency between all environments. This model can be useful if you're following certain deployment patterns or if you need to extend your services to a new geographic region. An example is the Deployment stamps pattern.

Disaster recovery. In some situations, where recovery time might not be time-sensitive, IaC configurations can be used as part of a disaster recovery plan. For example, if infrastructure needs to be recreated in a second region, your IaC configurations can be used to do so. You need to consider deployment time and restoring the state of your infrastructure and the infrastructure itself.

When you plan for disaster and recovery, ensure that your disaster recovery plans are fully tested and that they meet your Business metrics.

Better understand your cloud resources

IaC can also help you better understand the state of your cloud resources.

Audit changes. Changes to your IaC configurations will be version-controlled in the same way as your code, such as through Git's version history. Version control means you can review each change that has happened, and understand who made it and when. This approach can be helpful if you're trying to understand why a resource is configured a specific way.

Metadata. Many types of IaC configurations let you add metadata, like code comments, to help explain why something is done a particular way. If your organization has a culture of documenting your code, apply the same principles to your infrastructure code.

Keep everything together. It's common for a developer to work on features that require both code and infrastructure changes. By keeping infrastructure defined as code, you can group application and infrastructure code to understand the relationship between them better. For example, if you see a change to an IaC configuration on a feature branch or in a pull request, you'll have a clearer understanding of what that change relates to.

Better understand Azure itself. The Azure portal is a great way to provision and configure resources; but it often simplifies the underlying resource model used. Using IaC means that you gain a deeper understanding of what's happening in Azure and how to troubleshoot it if something isn't working correctly. For example, when creating a set of virtual machines in the Azure portal, some of the underlying resource creation is abstracted for the deployment process. When you use IaC, not only do you have explicit control over resource creation, little is abstracted from the process, which provides a richer understanding of what's deployed and how it's configured.

Categories of IaC tooling

You can use many declarative infrastructure deployment technologies with Azure. Deployment technologies fall into two main categories.

- **Imperative IaC** involves writing scripts in a language like Bash, PowerShell, C# script files, or Python. These scripts programmatically run a series of steps to create or modify your resources. When you use imperative deployments, it's up to you to manage things like dependency sequencing, error control, and resource updates.
- **Declarative IaC** involves writing a definition of how you want your environment to look; the tooling then figures out how to make this happen by inspecting your current state, comparing it to the target state you've requested, and applying the differences. For more information, see Use automation to redue effort and error.

There are great Azure tooling options for both models. Here we describe three of the commonly used declarative IaC technologies for Azure: ARM templates, Bicep, and Terraform.

Automate deployments with ARM Templates

Azure Resource Manager (ARM) Templates provide an Azure native infrastructure as code solution. ARM Templates are written in a language derived from JavaScript Object Notation (JSON), and they define the infrastructure and configurations for Azure deployments. An ARM template is declarative. You state what you want to deploy, provide configuration values, and the Azure engine takes care of making the necessary Azure REST API put requests. Other benefits you gain by using ARM templates for infrastructure deployments include:

- **Parallel resource deployment:** The Azure deployment engine sequences resource deployments based on defined dependencies. If dependencies don't exist between two resources, they're deployed at the same time.
- **Modular deployments:** ARM templates can be broken up into multiple template files for reusability and modularization.
- **Day one resource support:** ARM templates support all Azure resources and resource properties as they're released.
- **Extensibility:** Azure deployments can be extended by using deployment scripts and other automation solutions.
- **Validation:** Azure deployments are evaluated against a validation API to catch configuration mistakes.
- **Testing:** The ARM template test toolkit provides a static code analysis framework for testing ARM templates.
- **Change preview:** ARM template what-if lets you see what will be changed before deploying an ARM template.
- **Tooling:** Language service extensions are available for both Visual Studio Code and Visual Studio to help you author ARM templates.

The following example demonstrates a simple ARM template that deploys a single Azure Storage account. In this example, a single parameter is defined to take in a name for the storage account. Under the resources section, a storage account is defined, the *storageName* parameter is used to provide a name, and the storage account details are defined. See the included documentation for an in-depth explanation of the different sections and configurations for ARM templates.

JSON

```json
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-
01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageName": {
```

```
                "type": "string",
                "defaultValue": "newStorageAccount"
            }
        },
        "resources": [
            {
                "name": "[parameters('storageName')]",
                "type": "Microsoft.Storage/storageAccounts",
                "apiVersion": "2019-06-01",
                "location": "[resourceGroup().location]",
                "kind": "StorageV2",
                "sku": {
                    "name": "Premium_LRS",
                    "tier": "Premium"
                }
            }
        ]
    }
```

Learn more

- What are ARM templates?
- Deploy consistent infrastructure with ARM Templates
- ARM templates code samples

Automate deployments with Bicep

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. Bicep provides concise syntax, reliable type safety, and support for code reuse.

You can think of Bicep as a revision to the Azure Resource Manager template (ARM template) language rather than a new language. The syntax is different, but the core functionality and runtime remain the same.

The following example shows a simple Bicep file that defines a storage account.

Bicep

```
param location string = resourceGroup().location
param storageAccountName string =
'toylaunch${uniqueString(resourceGroup().id)}'

resource storageAccount 'Microsoft.Storage/storageAccounts@2021-06-01' = {
  name: storageAccountName
  location: location
  sku: {
    name: 'Standard_LRS'
  }
```

```
  kind: 'StorageV2'
  properties: {
    accessTier: 'Hot'
  }
}
```

Learn more

- Documentation: What is Bicep?
- Documentation: Frequently asked questions
- Documentation: Key Bicep concepts

Automate deployments with Terraform

Terraform is a declarative framework for deploying and configuring infrastructure that supports many private and public clouds, Azure being one of them. It has the main advantage of offering a cloud-agnostic framework. While Terraform configurations are specific to each cloud, the framework itself is the same for all of them. Terraform configurations are written in a domain-specific language (DSL) called Hashicorp Configuration Language.

The following example demonstrates a simple Terraform configuration that deploys an Azure resource group and a single Azure Storage account.

HashiCorp Configuration Language

```
resource "azurerm_resource_group" "example" {
  name     = "newStorageAccount"
  location = "eastus"
}

resource "azurerm_storage_account" "example" {
  name                     = "storageaccountname"
  resource_group_name      = azurerm_resource_group.example.name
  location                 = azurerm_resource_group.example.location
  account_tier             = "Standard"
  account_replication_type = "GRS"
}
```

Take note, the Terraform provider for Azure is an abstraction on top of Azure APIs. This abstraction is beneficial because the API complexities are obfuscated. This abstraction comes at a cost; the Terraform provider for Azure doesn't always provide parity with the Azure APIs' capabilities. To learn more about using Terraform on Azure, see Using Terraform on Azure

Automate infrastructure deployment with Azure Deployment Environments

For organizations that support multiple development teams working in multiple environments, it is essential to have repeatable, consistent, and secure environments. Development infrastructure administrators (dev infra admins) are responsible for managing and controlling access to the resources that developers need. Developers typically work on multiple projects, and might need several different development environments. Developers are usually organized in development teams, led by a development team lead.

Dev infra admin teams must be able to provide developers with environments that include the preconfigured resources needed for different environment types, like development, test, and production. Ideally, developers should be able to create a new instance of an environment appropriate for their project quickly and easily, rather than having to wait for the dev infra team.

By using project level templates, organizations can create environment templates that define the resources and configuration required for each project. Controlling access to each project enables the organization to set an identity security perimeter, so only approved admins can manage the project, and only approved developers can use it.

Creating environments by using templates helps organizations to avoid the pitfalls of manually creating each environment. Creating environments manually is time-consuming, often results in inconsistencies, and presents difficulties in ensuring compliance. When evaluating the use of templates, organizations should consider the overhead of creating new templates, maintaining existing templates, and retiring unused templates.

Azure Deployment Environments enables development teams deploy application infrastructure with project-based templates. Dev infra admins can deploy preconfigured resources defined by an IaC template. Environments can be platform as a service (PaaS) or infrastructure as a service (IaaS) environments based on the content of the template.

Dev infra admins can automate development infrastructure deployment by using Azure Deployment Environments. It automates the process of applying policies, permissions, and settings on environments, controlling the resource configuration that developers can create, and centrally tracking environments across projects by:

- Providing a project-based, curated set of reusable IaC templates.
- Defining specific Azure deployment configurations per project and per environment type.

- Providing self-service experience without giving control over subscriptions.
- Tracking costs and ensuring compliance with enterprise governance policies.

Learn more

- Azure Deployment Environments

Manual deployment

Manual deployment steps introduce significant risks regarding human error and also increase overall deployment times. However, in some cases, manual steps might be required. For these cases, ensure that any manual steps are documented, including roles and responsibilities.

Hotfix process

In some cases, you might have an unplanned deployment need. For instance, to deploy critical hotfixes or security remediation patches. A defined process for unplanned deployments can help prevent service availability and other deployment issues during these critical events.

Next steps

Automate infrastructure configuration

Configure infrastructure

Article • 05/03/2023

When you work with Azure, many services can be created and configured programmatically by using automation or infrastructure as code tooling. These tools access Azure through the exposed REST APIs or what we refer to as the Azure control plane. For example, an Azure Network Security Group can be deployed, and security group rules created using an Azure Resource Manager template. The Network Security Group and its configuration are exposed through the Azure control plane and natively accessible.

Other configurations, such as installing software on a virtual machine (VM), adding data to a database, or starting pods in an Azure Kubernetes Service cluster can't be accessed through the Azure control plane. These actions require a different set of configuration tools. These configurations are on the Azure data plane side, or not exposed through Azure REST APIs. The data plane enables tools to use agents, networking, or other access methods to provide resource-specific configuration options.

For example, when deploying a set of VMs to Azure, you might also want to install and configure a web server, stage the content, and then make the content available on the internet. Also, if the VM configuration changes and no longer aligns with the configuration definition, you might want a configuration management system to remediate the configuration. Many options are available for these data plane configurations. This article details several options and provides links for in-depth information.

Bootstrap automation

When deploying to Azure, you might need to run post-deployment VM configuration or run other arbitrary code to bootstrap the deployed Azure resources. Several options are available for these bootstrapping tasks and are detailed in the following sections of this article.

Azure VM extensions

Azure VM extensions are small packages that run post-deployment configuration and automation on Azure VMs. Several extensions are available for many different configuration tasks, such as running scripts, configuring antimalware solutions, and configuring logging solutions. These extensions can be installed and run on VMs by using an ARM template, the Azure CLI, Azure PowerShell module, or the Azure portal.

Each Azure VM has a VM Agent installed, and this agent manages the lifecycle of the extension.

A typical VM extension use case is to use a custom script extension to install software, run commands, and perform configurations on a virtual machine or Azure Virtual Machine Scale Sets. The custom script extension uses the Azure VM Agent to download and run a script. The custom script extensions can be configured to run as part of infrastructure as code deployments such that the VM is created, and then the script extension is run on the VM. Extensions can also be run outside of an Azure deployment by using the Azure CLI, PowerShell module, or the Azure portal.

In the following example, the Azure CLI is used to deploy a custom script extension to an existing virtual machine, which installs a Nginx webserver.

```
az vm extension set \
  --resource-group myResourceGroup \
  --vm-name myVM --name customScript \
  --publisher Microsoft.Azure.Extensions \
  --settings '{"commandToExecute": "apt-get install -y nginx"}'
```

Learn more

Use the included code sample to deploy a virtual machine and configure a web server on that machine with the custom script extension. For more information, see:

- Azure virtual machine extensions and features
- Azure Well Architected Framework sample (custom script extension)

cloud-init

cloud-init is a known industry tool for configuring Linux virtual machines on first boot. Much like the Azure custom script extension, cloud-init lets you install packages and run commands on Linux virtual machines. cloud-init can be used for things like software installation, system configurations, and content staging. Azure includes many cloud-init enabled Marketplace virtual machine images across many of the most well-known Linux distributions. For a full list, see cloud-init support for virtual machines in Azure.

1. To use cloud-init, create a text file named *cloud-init.txt* and enter your cloud-init configuration. In this example, the Nginx package is added to the cloud-init configuration.

YAML

```
#cloud-config
package_upgrade: true
packages:
  - nginx
```

2. Create a resource group for the VM.

Azure CLI

```
az group create --name myResourceGroupAutomate --location eastus
```

3. Create the VM, specifying the *--custom-data* property with the cloud-init configuration name.

Azure CLI

```
az vm create \
    --resource-group myResourceGroupAutomate \
    --name myAutomatedVM \
    --image UbuntuLTS \
    --admin-username azureuser \
    --generate-ssh-keys \
    --custom-data cloud-init.txt
```

On boot, cloud-init uses the system's native package management tool to install Nginx.

Learn more

- cloud-init support for virtual machines in Azure

Azure deployment script resource

When you perform Azure deployments, you might need to run arbitrary code for bootstrapping things like managing user accounts, Kubernetes pods, or querying data from a non-Azure system. Because none of these operations are accessible through the Azure control plane, some other mechanism is required for performing this automation. To run arbitrary code with an Azure deployment, see the `Microsoft.Resources/deploymentScripts` Azure resource.

The deployment script resource behaves like any other Azure resource in the following ways:

- Can be used in an ARM template.
- Contains ARM template dependencies on other resources.
- Consumes input, produces output.

- Uses a user-assigned managed identity for authentication.

When deployed, the deployment script runs PowerShell or Azure CLI commands and scripts. Script execution and logging can be observed in the Azure portal or with the Azure CLI and PowerShell module. Many options can be configured like environment variables for the execution environment, timeout options, and what to do with the resource after a script failure.

The following example shows an ARM template snippet with the deployment script resource configured to run a PowerShell script.

JSON

```json
{
    "type": "Microsoft.Resources/deploymentScripts",
    "apiVersion": "2019-10-01-preview",
    "name": "runPowerShellScript",
    "location": "[resourceGroup().location]",
    "kind": "AzurePowerShell",
    "identity": {
        "type": "UserAssigned",
        "userAssignedIdentities": {"[parameters('identity')]": {}}
    },
    "properties": {
        "forceUpdateTag": "1",
        "azPowerShellVersion": "3.0",
        "arguments": "[concat('-sqlServer ', parameters('sqlServer'))]",
        "primaryScriptUri": "[variables('script')]",
        "timeout": "PT30M",
        "cleanupPreference": "OnSuccess",
        "retentionInterval": "P1D"
    }
}
```

Learn more

- Documentation: Use deployment scripts in templates

Configuration Management

Configuration management tools can be used to configure and manage the ongoing configuration and state of Azure virtual machines. Three popular options are Azure Automation State Configuration, Chef, and Puppet.

Azure Automation State Configuration

Azure Automation State Configuration is a configuration management solution built on top of PowerShell Desired State Configuration (DSC). State configuration works with Azure virtual machines, on-premises machines, and machines in a cloud other than Azure. When you use state configuration, you can import PowerShell DSC resources and assign them to many virtual machines from a central location. Once each endpoint has evaluated and applied the desired state, state compliance is reported to Azure and can be seen on a built-in dashboard.

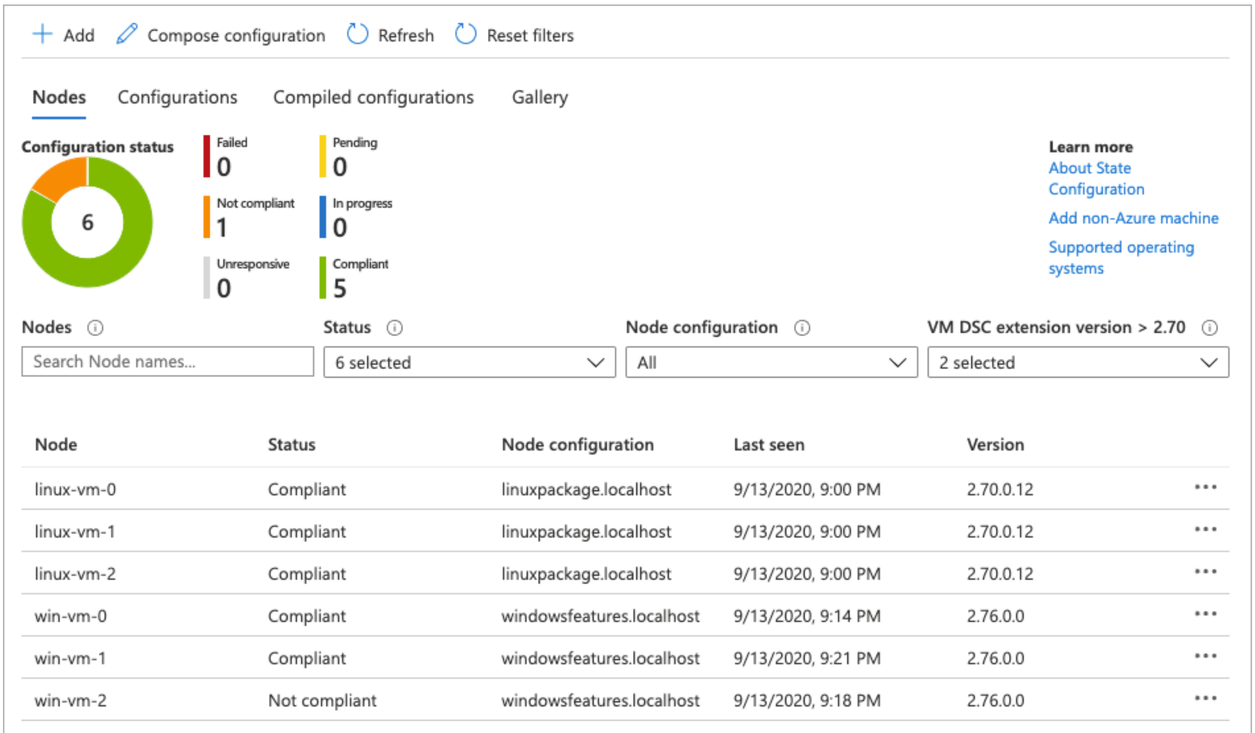
The following example uses PowerShell DSC to ensure the NGINX has been installed on Linux systems.

```PowerShell
configuration linuxpackage {

    Import-DSCResource -Module nx

    Node "localhost" {
        nxPackage nginx {
            Name = "nginx"
            Ensure = "Present"
        }
    }
}
```

Once the module imports into Azure Automation State Configuration and is assigned to nodes, the state configuration dashboard provides compliance results.



Learn more

Use the included code sample to deploy Azure Automation State Configuration and several Azure virtual machines. The virtual machines are also onboarded to state configuration, and a configuration applied.

- Get started with Azure Automation State Configuration
- Azure Automation State Configuration example scenario

Chef

Chef is an automation platform that helps define how your infrastructure is configured, deployed, and managed. Other components included in Chef Habitat are for application lifecycle automation, rather than the infrastructure and Chef InSpec that helps automate compliance with security and policy requirements. Chef Clients are installed on target machines, with one or more central Chef Servers that store and manage the configurations.

Learn more

- Documentation: An Overview of Chef

Puppet

Puppet is an enterprise-ready automation platform that handles the application delivery and deployment process. Agents are installed on target machines to let Puppet run manifests that define the desired configuration of the Azure infrastructure and virtual machines. Puppet can integrate with other solutions such as Jenkins and GitHub for an improved DevOps workflow.

Learn more

- Documentation: How Puppet works

Next steps

Automate operational tasks

Automate operational tasks

Article • 05/01/2023

Operational tasks can include any action or activity you may perform while managing systems, system access, and processes. Some examples include rebooting servers, creating accounts, and shipping logs to a data store. These tasks may occur on a schedule, as a response to an event or monitoring alert, or ad-hoc based on external factors. Like many other activities related to managing computer systems, these activities are often performed manually, which takes time, and are error-prone.

Many of these operational tasks can and should be automated. Using scripting technologies and related solutions, you can shift effort from manually performing operational tasks towards building automation for these tasks. In doing so, you achieve so much, including:

- Reduce time to perform an action.
- Reduce risk in performing the action.
- Automated response to events and alerts.
- Increased human capacity for further innovation.

When working in Azure, you have many options for automating operational tasks. This document details some of the more popular.

Azure Functions

Azure Functions allows you to run code without managing the underlying infrastructure on where the code is run. Functions provide a cost-effective, scalable, and event-driven platform for building applications and running operational tasks. Functions support running code written in a range of programming languages including C#, Java, JavaScript, Python, and PowerShell.

When creating a Function, a hosting plan is selected. Hosting plans controls how a function app is scaled, resource availability, and availability of advanced features such as virtual network connectivity and startup time. The hosting plan also influences the cost.

Functions hosting plans:

- **Consumption:** Default hosting plan, pay only for Function execution time, configurable timeout period, automatic scale.
- **Premium:** Faster start, VNet connectivity, unlimited execution duration, premium instance sizes, more predictable pricing.

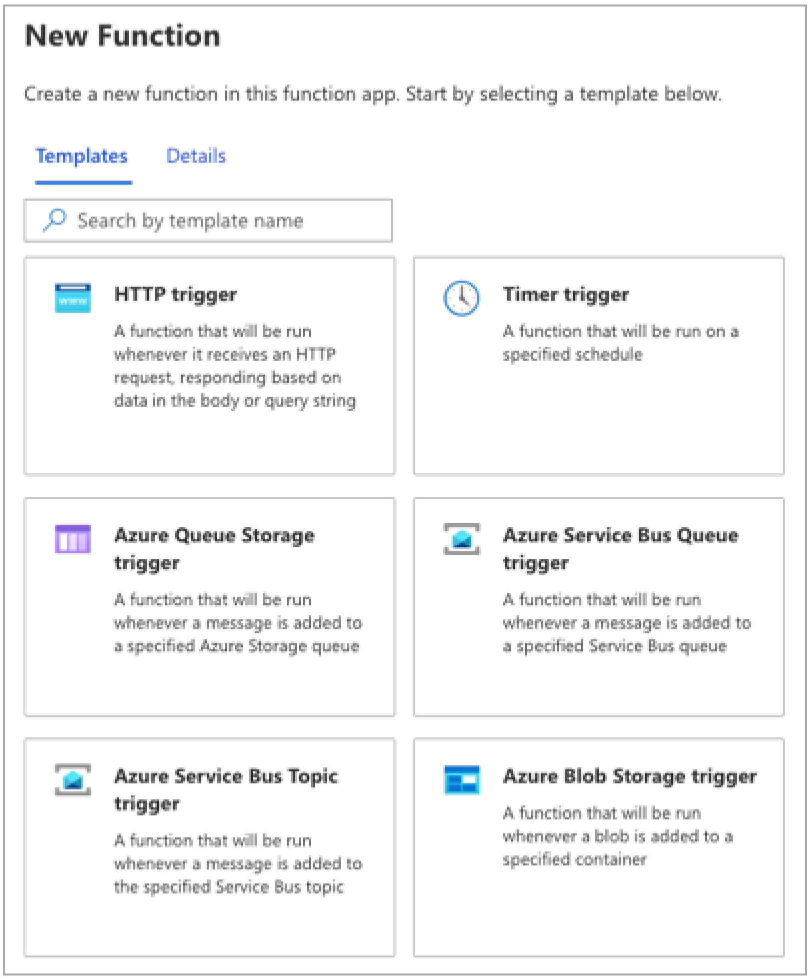
- **Dedicated:** Functions run on dedicated virtual machines and can use custom images.

For full details on consumption plans, see Azure Functions scale and hosting.

Functions provide event-driven automation; each function has a trigger associated with it. These triggers are what run the functions. Common triggers include:

- **HTTP / Webhook:** Function is run when an HTTP request is received.
- **Queue:** Function is run when a new message arrives in a message queue.
- **Blob storage:** Function is run when a new blob is created in a storage container.
- **Timer:** Function is run on a schedule.

Below are example triggers seen in the Azure portal when creating a new function



Once an event has occurred that initiates a Function, you may also want to consume data from this event or from another source. Once a Function has been completed, you may want to publish or push data to an Azure service such as a Blob Storage. Input and output are achieved using input and output bindings. For more information about triggers and bindings, see Azure Functions triggers and binding concepts.

Both PowerShell and Python are common languages for automating everyday operational tasks. Because Azure Functions supports both of these languages, it is an excellent platform for hosting, running, and auditing automated operational tasks. For example, let's assume that you would like to build a solution to facilitate self-service account creation. An Azure PowerShell Function could be used to create the account in Azure Active Directory. An HTTP trigger can be used to initiate the Function, and an input binding configured to pull the account details from the HTTP request body. The only remaining piece would be a solution that consumes the account details and creates the HTTP requests against the Function.

Learn more

- Documentation: Azure Functions PowerShell developer guide
- Documentation: Azure Functions Python developer guide

Azure Deployment Environments

Azure Deployment Environments enables development teams to quickly spin up consistent app infrastructure by using project-based templates, minimizing setup time while maximizing security, compliance, and cost efficiency. A deployment environment is a collection of Azure resources deployed in predefined subscriptions. Development infrastructure (dev infra) admins can enforce enterprise security policies and provide a curated set of predefined infrastructure as code (IaC) templates.

Dev infra admins define deployment environments as catalog items hosted in a GitHub or Azure DevOps repository called a catalog. A catalog item consists of an IaC template and a manifest.yaml file.

The creation of deployment environments can be scripted, and the environments can be managed programmatically.

Learn more

- Azure Deployment Environments
- Scenarios for using Azure Deployment Environments Preview
- Create and access an environment by using the Azure CLI

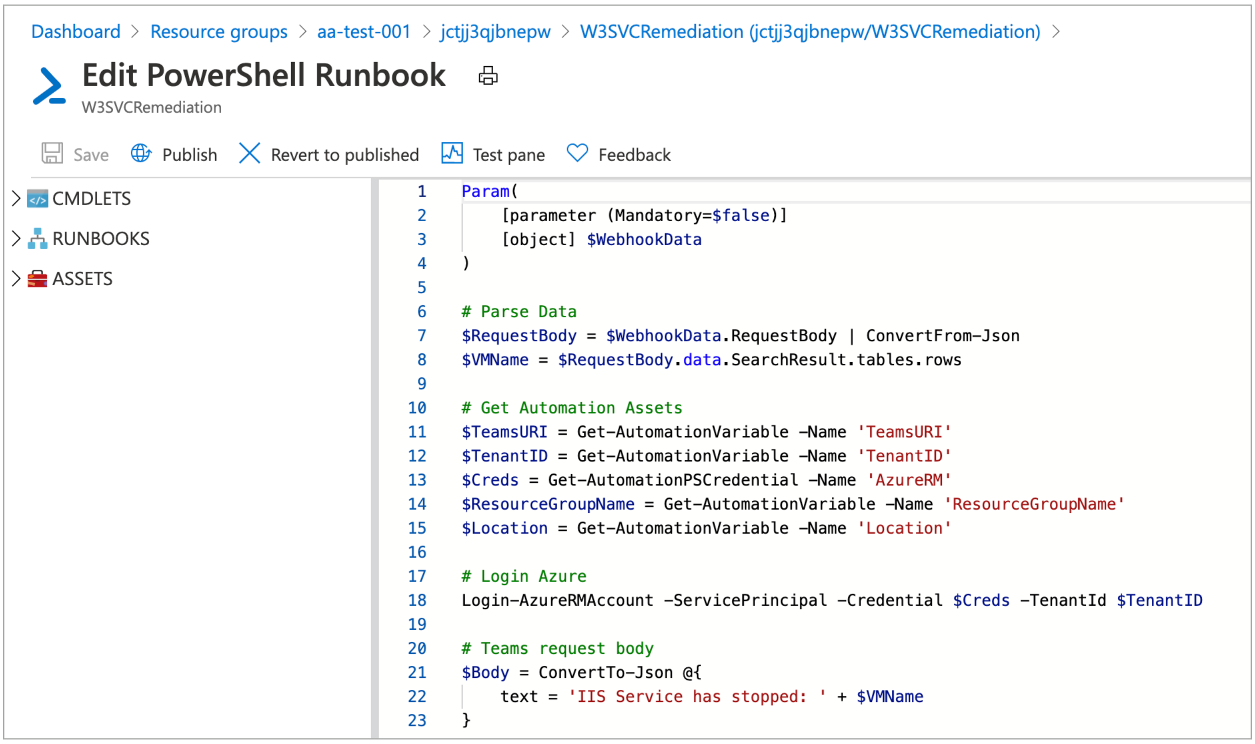
Azure Automation

PowerShell and Python are popular programming languages for automating operational tasks. Using these languages, performing operations like restarting services, moving logs between data stores, and scaling infrastructure to meet demand can be expressed in code and executed on demand. Alone, these languages do not offer a platform for centralized management, version control, or execution history. The languages also lack a native mechanism for responding to events like monitoring driven alerts. To provide these capabilities, an automation platform is needed.

Azure Automation provides an Azure-hosted platform for hosting and running PowerShell and Python code across Azure, non-Azure cloud, and on-premises environments. PowerShell and Python code is stored in an Azure Automation Runbook, which has the following attributes:

- Execute Runbooks on demand, on a schedule, or through a webhook.
- Execution history and logging.
- Integrated secrets store.
- Source Control integration.

As seen in the following image, Azure Automation provides a portal experience for managing Azure Automation Runbooks. Use the included code sample (ARM template) to deploy an Azure Automation account, automation runbook, and explore Azure Automation for yourself.



Learn more

- Documentation: What is Azure Automation?

Scale operations

So far, this document has detailed options for scripting operational tasks; however, many Azure services come with built-in automation, particularly in scale operations. As application demand increases (transactions, memory consumption, and compute availability), you may need to scale the application hosting platform so that requests continue to be served. As demand decreases, scaling back not only appropriately resizes your application but also reduces operational cost.

In cloud computing, scale activities are classified into two buckets:

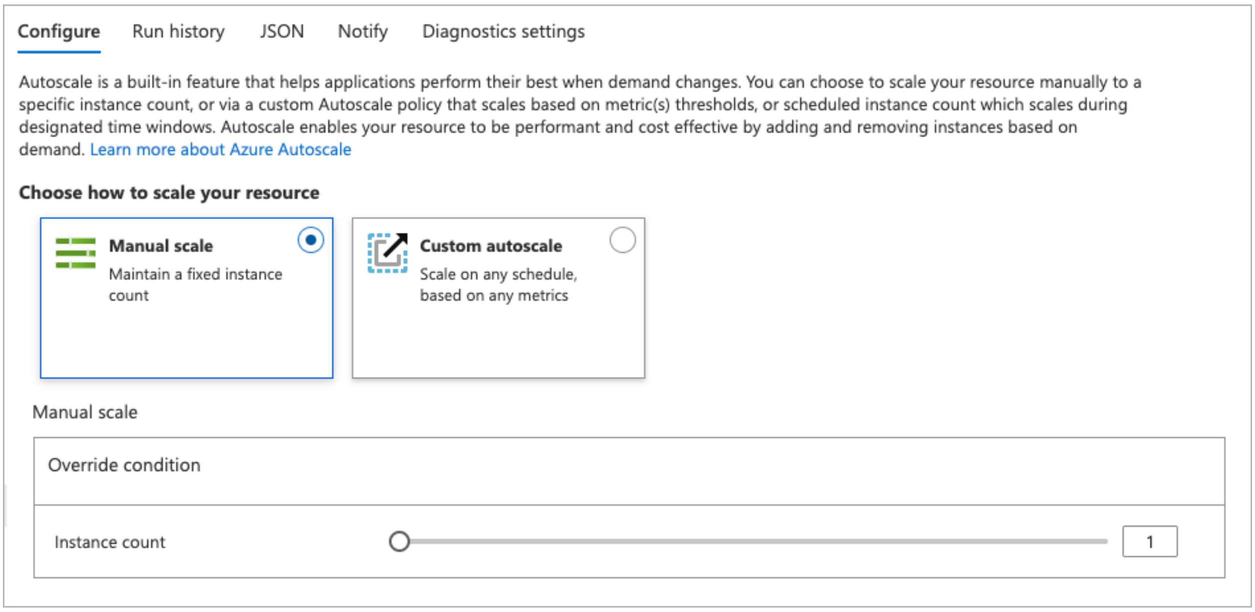
- **Scale-up:** Adding extra resources to an existing system to meet demand.
- **Scale-out:** Adding more infrastructure to meet demand.

Many Azure services can be scaled up by changing the pricing tier of that service. Generally, this operation would need to be performed manually or using detection logic and custom automation.

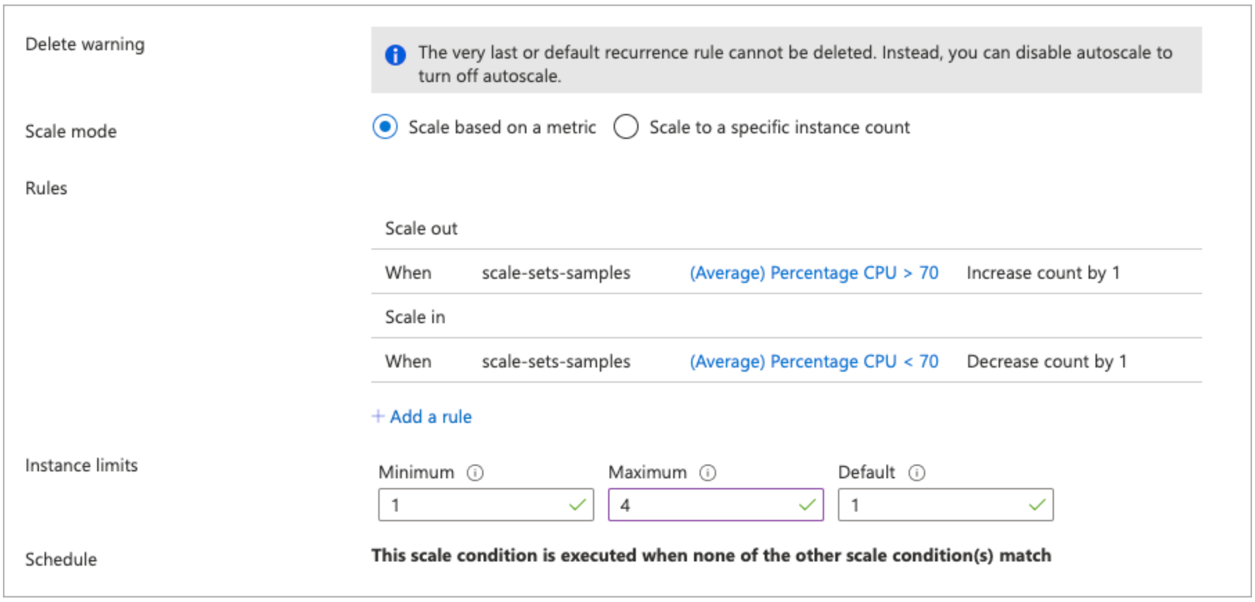
Some Azure services support automatic scale-out, which is the focus of this section.

Azure Monitor autoscale

Azure Monitor autoscale can be used to autoscale Virtual Machine Scale Sets, Cloud Services, App Service Web Apps, and API Management service. To configure scale-out operations for these services, while in the Azure portal, select the service, and then **scale-out** under the resource settings. Select **Custom** To configure autoscaling rules. Automatic scale operations can also be configured using an Azure Resource Manager Template, the Azure PowerShell module, and the Azure CLI.



When creating the autoscale rules, configure minimum and maximum instance counts. These settings prevent inadvertent costly scale operations. Next, configure autoscale rules, at minimum one to add more instances, and one to remove instances when no longer needed. Azure Monitor autoscale rules give you fine-grain control over when a scale operation is initiated. See the Learn more section below for more information on configuring these rules.



Learn more

- Documentation: Azure Monitor autoscale overview

Azure Kubernetes Service

Azure Kubernetes Service (AKS) offers an Azure managed Kubernetes cluster. When considering scale operations in Kubernetes, there are two components:

- **Pod scaling:** Increase or decrease the number of load balanced pods to meet application demand.
- **Node scaling:** Increase or decrease the number of cluster nodes to meet cluster demand.

Azure Kubernetes Service includes automation to facilitate both of these scale operation types.

Horizontal pod autoscaler

Horizontal pod autoscaler (HPA) monitors resource demand and automatically scales pod replicas. When configuring horizontal pod autoscaling, you provide the minimum and maximum pod replicas that a cluster can run and the metrics and thresholds that initiate the scale operation. To use horizontal pod autoscaling, each pod must be configured with resource requests and limits, and a **HorizontalPodAutoscaler** Kubernetes object must be created.

The following Kubernetes manifest demonstrates resource requests on a Kubernetes pod and also the definition of a horizontal pod autoscaler object.

YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-vote-back
spec:
  replicas: 1
  selector:
    matchLabels:
      app: azure-vote-back
  template:
    metadata:
      labels:
        app: azure-vote-back
    spec:
      nodeSelector:
        "beta.kubernetes.io/os": linux
      containers:
      - name: azure-vote-back
        image: redis
        # Resource requests and limits
        resources:
          requests:
            cpu: 100m
            memory: 128Mi
          limits:
            cpu: 250m
            memory: 256Mi
        ports:
        - containerPort: 6379
          name: redis
---
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: azure-vote-back-hpa
spec:
  maxReplicas: 10 # define max replica count
  minReplicas: 3  # define min replica count
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: azure-vote-back
  targetCPUUtilizationPercentage: 50 # target CPU utilization
```

Cluster autoscaler

Where horizontal pod autoscaling is a response to demand on a specific application of service running in a Kubernetes cluster, cluster autoscaling responds to demand on the entire Kubernetes cluster itself. If a Kubernetes cluster does not have enough compute resources or nodes to facilitate all requested pods' resource requests, some of these pods will enter a non-scheduled or pending state. In response to this situation, more nodes can be automatically added to the cluster. Conversely, once compute resources have been freed up, the cluster nodes can automatically be removed to match steady-state demand.

Cluster autoscaler can be configured when creating an AKS cluster. The following example demonstrates this operation with the Azure CLI. This operation can also be completed with an Azure Resource Manager template.

Azure CLI

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 1 \
  --vm-set-type VirtualMachineScaleSets \
  --load-balancer-sku standard \
  --enable-cluster-autoscaler \
  --min-count 1 \
  --max-count 3
```

Cluster autoscaler can also be configured on an existing cluster using the following Azure CLI command.

Azure CLI

```
az aks update \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --enable-cluster-autoscaler \
  --min-count 1 \
  --max-count 3
```

See the included documentation for information on fine-grain control for cluster autoscale operations.

Learn more

- Documentation: Horizontal pod autoscaling
- Documentation: AKS cluster autoscaler

Release engineering: Application development

Article • 05/18/2023

One of the primary goals of adopting modern release management strategies is to build systems that let your teams turn ideas into production-delivered software with as little friction as possible. Throughout this section of the Well-Architected Framework, methods and tools for quickly and reliably delivering software are examined. You'll learn about continuous deployment, integration strategies, deployment environments, and more. Samples are provided to help you quickly get hands-on with this technology.

However, release engineering doesn't start with fancy deployment software, multiple deployment environments, or Kubernetes clusters. Before examining how we can quickly and reliably release software, we need to first look at how software is developed.

Not only has the introduction of cloud computing affected how software is delivered and run, it has also had a huge downstream effect on how software is developed. For example, the introduction of container technology has changed how we can host, scale, and deprecate software. That said, containers have also impacted dependency management, host environment, and tooling as we develop software.

This article details many practices to consider when building strategies for developing for the cloud, including:

- Development environments, or where you write your code.
- Source control and branching strategies, or how you manage, collaborate on, and eventually deploy your code.

Development environments

When you develop software for the cloud or any other environment, care needs to be taken to ensure that the development environment is set up for success. When setting up a development environment, consider the following questions:

- How do I ensure that all dependencies are in place?
- How can I best configure my development environment to emulate a production environment?
- How do I develop code where service dependencies might exist with code already in production?

Based on the chosen deployment method and target application, the answers to these questions, and the tools used for what's often referred to the "inner-loop" development process, can differ.

Package management solutions, for example, can help you manage dependencies in your development environment. When you use containers, you can use a solution like Docker Desktop for your development environment instead. When it comes to effectively emulating a production environment, you can choose to set up a range of dummy services locally.

Some Linux tools are a handy solution, and they're available on Windows machines too. You can use them by configuring the Windows Subsystem for Linux.

Other tools let you provide developers with a preconfigured environment that they can spin up whenever they need it, providing a consistent environment. Azure Deployment Environments helps you to emulate environments, such as sandbox, testing, staging, or production.

Finally, tools like Bridge for Kubernetes let you debug code in your development environment while being connected to a Kubernetes cluster. This configuration can be helpful when working on containerized microservices. You can work on one service locally while services that you take a dependency on are spun up remotely.

Examples of inner-loop development tools

A development team might choose some of the following pieces of software when developing a containerized application with a Kubernetes cluster as its deployment target.

- Azure Artifacts is a package management solution that lets you host private packages and provide upstream connectivity to public package repositories for various package management systems such as NuGet.
- Docker Desktop is an application that provides a Docker environment on your development system. Docker Desktop includes not only the Docker runtime but also application development tools and local Kubernetes environments.
- Windows Subsystem for Linux provides a Linux environment on your Windows machines, including many command-line tools, utilities, and Linux applications.
- Bridge to Kubernetes lets you run and debug code on your development system while connected to a Kubernetes cluster. This configuration is helpful when working on microservice type architectures.
- Podman is an open-source tool for working with containers.

- Microsoft Dev Box provides a safe development environment for developers to create code, supported through virtual networks and RBAC permissions that control who can access the organizations network and who can use the dev box.
- Azure deployment environments empowers development teams to quickly and easily spin up app infrastructure with project-based templates that establish consistency and best practices while maximizing security.

Source control

Source control management (SCM) systems provide a way to control, collaborate, and peer review software changes. As software is merged into source control, the system helps manage code conflicts. Ultimately, source control provides a running history of the software, modifications, and contributors. Whether a piece of software is open-sourced or private, the use of source control software has become a standardized method of managing software development.

As cloud practices are adopted, and because so much of the cloud infrastructure is managed through code, version control systems are an integral part of infrastructure management.

Many source control systems are powered by Git. Git is a distributed version control system with related tools that lets you and your team track source code changes during the software development lifecycle. When you use Git, you can create a copy of the software, make changes, propose the changes, and receive peer review on your proposal. Modern source control software simplifies this peer review process and helps you see the exact changes that a contributor wants to make.

Once the proposed changes have been approved, Git helps merge the changes into the source, including conflict resolution. If at any point the changes need to be reverted, Git can also manage rollback.

Version control and code changes

Beyond providing a place to store code, source control systems let you understand what version of the software is current and to identify changes between the present and past versions. Version control solutions also provide a method for reverting to the previous version when needed.

The following image demonstrates how Git and GitHub are used to see the proposed code changes.

Branches, forks and pull requests

When you use source control systems, you can create branches in an existing repo and your own copies of the software. A copy of an existing repo is called a "fork".

While branches are usually short lived and will eventually be merged back into a repo via a pull request, forks usually remain for more time. While they can also be merged back via a pull request, some forks develop independently from their origin repo.

You can review our documentation on Forks and Branches to find out more about the differences and similarities between them.

Peer review

As updates are made to software and infrastructure configurations, version control software lets us propose these changes before merging them into the source. During the proposal, peers can review the changes, recommend updates, and approve the changes. Source control solutions provide an excellent platform for collaboration on changes to the software.

To learn more about Git, see What is Git?

GitHub

GitHub is a popular source control system that uses Git. In addition to core Git functionality, GitHub includes features such as access control, collaboration features such as code issues, project boards, wikis, and an automation platform called GitHub actions.

Azure Repos

Azure DevOps is a collection of services for building, collaborating on, testing, and delivering software to any environment. Azure DevOps Services includes Azure Repos, which is its source control system. When you use Azure Repos, you also receive unlimited free private Git repositories. Standard Git powers Azure Repos, and you can use clients and tools of your choice for working with them.

Next steps

Release Engineering: Continuous integration

Release engineering: Continuous integration

Article • 05/18/2023

As code is developed, updated, or even removed, having a friction-free and safe method to integrate these changes into the main code branch is paramount to enabling developers to provide value fast. As a developer, you can make small code changes, push these changes to a code repository, and get almost instantaneous feedback on the quality, test coverage, and introduced bugs. This process lets you work faster and with more confidence and less risk. Continuous integration (CI) is a practice where source control systems and software deployment pipelines are integrated to provide automated build, test, and feedback mechanisms for software development teams.

Continuous integration is about ensuring that software is ready for deployment but doesn't include the deployment itself. This article covers the basics of continuous integration and offers links and examples for more in-depth content.

Continuous integration

Continuous integration is a software development practice with which developers integrate software updates into a source control system on a regular cadence. The continuous integration process starts when an engineer creates a pull request signaling to the CI system that code changes are ready to be integrated. Ideally, integration validates the code against several baselines and tests and provides quick feedback to the requesting engineer on the status of these tests. Assuming baseline checks and testing have gone well, the integration process produces and stages assets such as compiled code and container images that will deploy the updated software.

As a software engineer, continuous integration can help you deliver quality software more quickly by performing the following actions:

- Run automated tests against the code, providing early detection of breaking changes.
- Run code analysis to ensure code standards, quality, and configuration.
- Run compliance and security checks ensuring no known vulnerabilities.
- Run acceptance or functional tested to ensure that the software operates as expected.
- To provide quick feedback on detected issues.
- Where applicable, produce deployable assets or packages that include the updated code.

To achieve continuous integration, use software solutions to manage, integrate, and automate the process. A common practice is to use a continuous integration pipeline, detailed in this article's next section.

Continuous integration pipelines

A continuous integration pipeline involves a piece of software, in many cases cloud-hosted, that provides:

- A platform for running automated tests
- Compliance scans
- Reporting
- All other components that make up the continuous integration process

In most cases, the pipeline software is attached to source control such that when pull requests are created or software is merged into a specific branch, the continuous integration pipeline is run. Source control integration also provides the opportunity for providing CI feedback directly on pull requests.

Many solutions provide continuous integration pipeline capabilities.

Learn more

To learn how to create a continuous integration pipeline with either GitHub or Azure DevOps, see:

- Create your first pipeline
- Using starter workflows

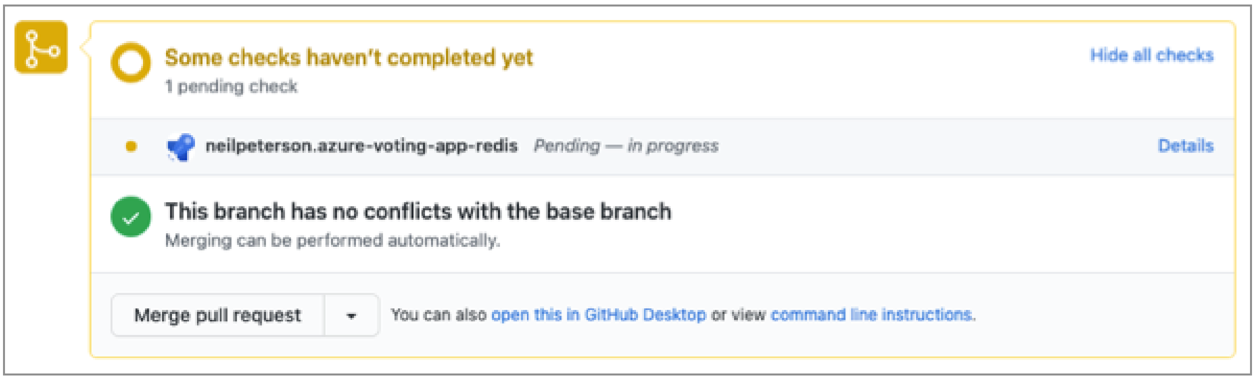
Source control integration

The integration of your continuous integration pipeline with your source control system is key to enabling fast, self-service, and friction-free code contributions.

- As pull requests are created, the CI pipeline is run, including all tests, security assessments, and other checks.
- CI test results are provided to the pull request initiator directly in the pull request, allowing for almost real-time feedback on quality.
- Another popular practice is building small reports or badges that can be presented in source control to make visible the current builds states

The following image shows the integration between GitHub and an Azure DevOps pipeline. In this example, a pull request has been created, which in turn has triggered an

Azure DevOps pipeline. The pipeline status can be seen directly in the pull request.



Test integration

A key element of continuous integration is the continual building and testing of code as code contributions are made. Testing pull requests as they're created gives quick feedback that the commit hasn't introduced breaking changes. The advantage is that the tests that are run by the continuous integration pipeline can be the same tests run during test-driven development.

The following code snippet shows a test step from an Azure DevOps pipeline. There are two actions occurring:

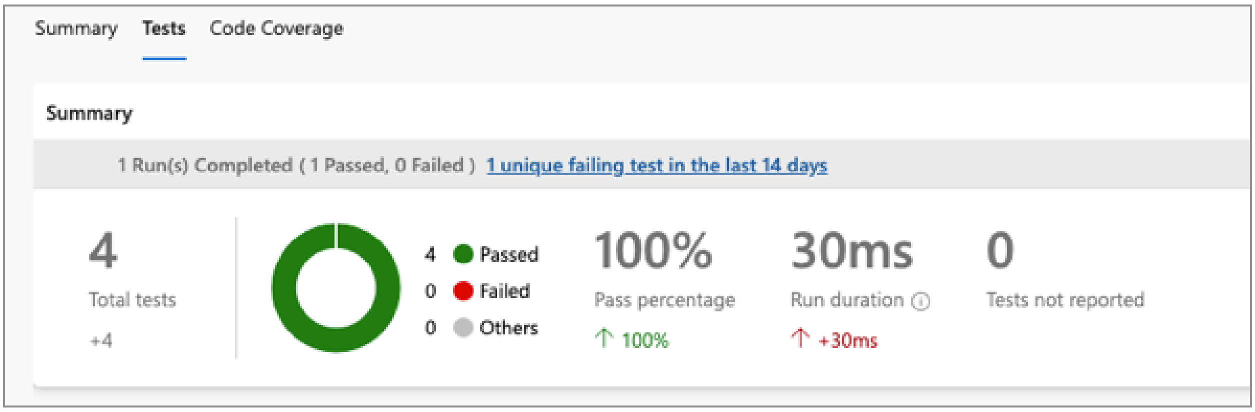
- The first task uses a popular Python testing framework to run CI tests. These tests reside in source control alongside the Python code. The test results are output to a file named *test-results.xml*.
- The second task consumes the test results and publishing them to the Azure DevOps pipeline as an integrated report.

YAML

```yaml
- script: |
    pip3 install pytest
    pytest azure-vote/azure-vote/tests/ --junitxml=junit/test-results.xml
    continueOnError: true

- task: PublishTestResults@2
    displayName: 'Publish Test Results'
    inputs:
    testResultsFormat: 'JUnit'
    testResultsFiles: '**/test-results.xml'
    failTaskOnFailedTests: true
    testRunTitle: 'Python $(python.version)'
```

The following image shows the test results as seen in the Azure DevOps portal:

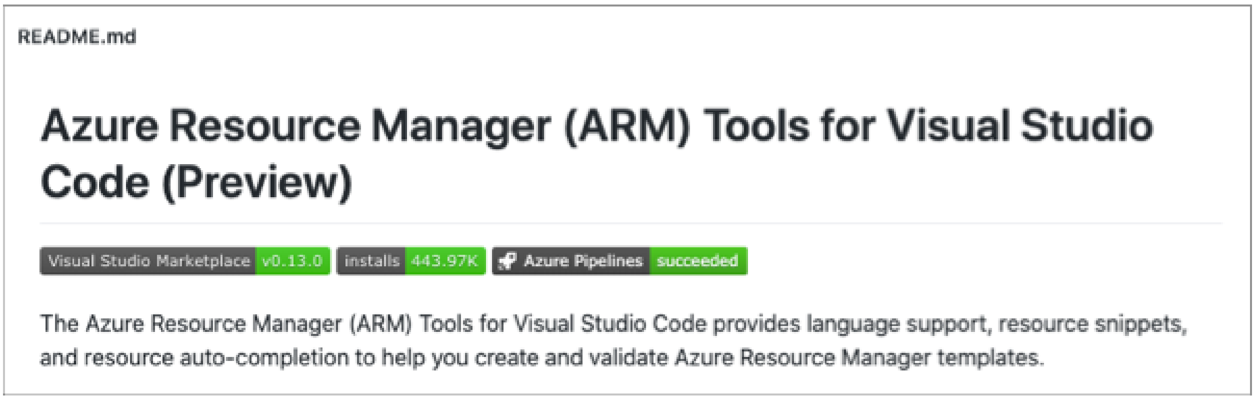


Failed tests

Failed tests should temporarily block a deployment and lead to a deeper analysis of what has happened. Failed tests should also lead to either a refinement of the test or an improvement of the change that caused the test to fail.

CI result badges

Many developers show that their code quality is high by displaying a status badge in their repo. The following image shows an Azure Pipelines badge as displayed on the Readme file for an open-source project on GitHub:



Learn more

To learn how to display badges in your repositories, see:

- Add an Azure Pipeline status badge to your repository.
- Add a GitHub workflow status badge to your repository .

Next steps

Release Engineering: Testing

Testing your application and Azure environment

Article • 05/30/2023

Testing is a fundamental component in DevOps and agile development in general. If automation gives DevOps the required speed and agility to deploy software quickly, then only the extensive testing of those deployments achieves the required reliability that customers demand.

A main tenet of system reliability is the "shift-left" principle. If developing and deploying an application is a process depicted as a series of steps going from left to right, testing shouldn't be limited to the very end of the process. It should be shifted as much to the beginning, meaning to the left, as possible. Errors are cheaper to repair when caught early. They can be expensive or impossible to fix later in the application life cycle.

Testing should occur on all code, including application code, infrastructure templates, and configuration scripts. As described in Repeatable infrastructure, the environment where applications are running should be version-controlled and deployed through the same mechanisms as application code. The environment can then be tested and validated by using the same testing paradigms that teams already use for application code.

A range of testing tools are available to automate and streamline different kinds of testing. These tools include Azure Load Testing, Azure Pipelines for automated testing and Azure Test Plans for manual testing.

There are multiple stages at which tests can be performed in the life cycle of code, and each of them has particularities that are important to understand. In this guide, you can find a summary of the different tests that you should consider while developing and deploying applications.

Automated testing

Automating tests is the best way to make sure that they're used consistently. Depending on how frequently tests are performed, they're typically limited in duration and scope, as the following different types of automated tests show.

Unit testing

Unit tests are tests typically run as part of the continuous integration routine. Unit Tests should be extensive and quick, ideally covering 100% of the code, and running in under 30 seconds.

Unit testing can verify that the syntax and functionality of individual modules of code are working the way they should, for example, producing a defined output for a known input. Unit tests can also be used to verify that infrastructure as code assets are valid.

Unit tests should be applied to all code assets, including templates and scripts.

Smoke testing

Smoke tests verify that a workload can be stood up in a test environment and performs as expected. Smoke tests don't go to the extent of integration tests as smoke tests don't verify the interoperability of different components.

Instead, they verify that the deployment methodology for both the infrastructure and the application works and that the system responds as intended once the process is complete.

Integration testing

After making sure that the different application components operate individually, integration testing determines whether they can interact with each other as they should.

Running a large integration test suite can take a considerable amount of time, which is why tests should be performed as early as possible (the shift-left principle) in the software development lifecycle. Integration tests should be reserved to scenarios that can't be tested with a smoke or unit test.

Long running test processes can be run on a regular interval if needed. A regular interval offers a good compromise, detecting interoperability issues between application components no later than one day after they were introduced.

Manual testing

Manual testing is more expensive than automated testing, so it's usually run less frequently. Manual testing is fundamental for the correct functioning of the DevOps feedback loop. It corrects errors before they become too expensive to repair or cause customer dissatisfaction.

Acceptance testing

Depending on the context, acceptance testing is sometimes performed manually. It can be partially or fully automated. Its purpose is to determine whether the software system has met the requirement specifications.

The main purpose of this test is to evaluate the system's compliance with the business requirements and verify if it has met the required criteria for delivery to end users.

Tools that include Usage analysis with Application Insights can help you work out how people use your application. These tools let you decide whether a new feature has improved your applications without bringing unintended adverse effects.

The next section of this article covers methods for testing in production, which can be used to verify if features have met business targets with real world user behavior.

Testing and experimentation in production

Depending on your chosen deployment strategy, Azure platform features might help you conduct experiments in production.

Azure AppService, for example, comes with the slot functionality to Set up staging environments that let you have two different versions of the same application running at the same time. Slots let you use A/B testing by directing part of the user traffic to one version of the application and the rest of the traffic to another.

There are multiple popular approaches to experimentation in production:

- **Blue/Green deployments:** When deploying a new application version, you can deploy it in parallel to the existing one. Deploying in parallel lets you start redirecting clients to the new version. If everything goes well, you can then decommission the old version. If there's any problem with the new deployment, you can redirect the users back to the deployment with the previous version.
- **Canary releases:** You can expose the new functionality of your application in a staggered way to select groups of users. You can extend the functionality to a larger group of users if:
  - Users are satisfied with the new functionality.
  - The functionality performs as expected with the control group.
- **A/B testing:** A/B testing is similar to canary release testing. While canary release testing focuses on mitigating risk, A/B testing focuses on evaluating two versions of an application or feature side by side. For example, if you have two versions of

the layout of your application, you could send half of your users to one, half to the other, and use metrics to see which layout is more successful.

Stress testing

As other sections of this framework have explained, designing your application code and infrastructure for scalability is of paramount importance. Stress tests help you verify regularly that your application and your environment adapt to changing load conditions.

During these stress tests, it's critical to monitor all the components of the system to identify potential bottlenecks and to establish a clear baseline to test against.

Every component of the system that isn't able to scale out can become a scale limitation, such as active or passive network components or databases. It's important to know the limits of components so that you can mitigate their effect when it comes to application scale.

Another important part of stress tests is to verify that the infrastructure scales back down to its normal condition in order to keep costs under control.

Business continuity testing

Business continuity testing offers opportunities for improving prevention and recovery methods for large-scale incidents. It also increases resilience to mundane incidents such as isolated hardware failure or data corruption.

Disaster recovery drills

Disaster recovery drills are important for verifying that the existing backup strategy is complimented by a working recovery procedure. These drills should be conducted regularly.

Tools such as Azure Site Recovery make it possible to start an isolated copy of the primary workload location in a secondary environment, so that it can be verified that the workload has recovered as it should in an emergency.

In case a problem occurs during the drill, the disaster recovery procedure can be optimized, and the infrastructure in the secondary environment can be deleted safely.

Exploratory testing

During exploratory testing, experts explore the application in its entirety trying to find faults or suboptimal implementations of functionality. These experts could be developers, UX specialists, product owners, actual users, and other profiles.

Structured test plans are typically not used since testing is left to the ability of the individual tester.

Fault injection

Fault injection tests if the application is resilient to infrastructure failures. The technique introduces faults in the underlying infrastructure and observes how the application behaves.

Fault injection testing is fundamental to build trust in your redundancy mechanisms. Shutting down infrastructure components, purposely degrading performance, or introducing faults are ways of verifying that the application is going to react as expected when these situations occur.

Products like Azure Chaos Studio aim to simplify the process of fault injection testing. Many larger organizations have built their own chaos engines that use automated testing tools to achieve fault injection.

Summary

In order to deploy software quickly and reliably, testing is a fundamental component of the development and deployment life cycle. It's important to make sure that the application is going to perform as expected in every situation. We therefore need to not only test application code but all aspects of our workload.

Next steps

Release Engineering: Performance

Performance considerations for your deployment infrastructure

Article • 05/30/2023

Build status shows if your product is in a deployable state, so builds are the heartbeat of your continuous delivery system. It's important to have a build process up and running the first day of your product development. Since builds provide such crucial information about the status of your product, you should always strive for fast builds.

It's difficult to fix a build problem if it takes longer to build. When delays happen and become normalized, teams tend to become less motivated to fix the problem.

Build times

Here are few ways you can achieve faster builds:

- **Choosing agents that meet your performance requirements:** Speeding up your builds starts with selecting the right build machines. Fast machines can make the difference between hours and minutes. If your pipelines are in Azure Pipelines, then you've got a convenient option to run your jobs by using a Microsoft-hosted agent. With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. For more information, see Microsoft-hosted agents.

- **Build server location:** When you're building your code, data is sent across the wire. Inputs to the builds are fetched from a source control repository and the artifact repository. At the end, the output from the build process needs to be copied, including not only the compiled artifacts, but also the test reports, the code coverage results, and the debug symbols. It's important that these copy actions are fast. If you use your own build server, ensure that the build server is located near the sources and a target location. Fast uploads and downloads can reduce the overall build time.

- **Scaling out build servers:** A single build server might be sufficient for a small product. As the size and the scope of the product and the number of teams working on the product increases, a single server might not be enough. Scale your infrastructure horizontally over multiple machines when you reach the limit. For more information, see Create and manage agent pools.

- **Optimizing the build:**

- Add parallel jobs to speed up the build process. For more information, see Configure and pay for parallel jobs.
- Enable parallel test suite runs, which often save a large amount of time, especially when running integration and UI tests. For more information, see Run tests in parallel for any test runner.
- Use the notion of a multiplier, where you can scale out your builds over multiple build agents. For more information, see Specify jobs in your pipeline.
- Consider moving integration, UI, and smoke tests to a release pipeline. Moving to a release pipeline improves the build speed and the speed of the build feedback loop.
- Publish the build artifacts to a package management solution, such as NuGet or Maven. Publishing to a package management solution lets you reuse your build artifact more easily.

Human intervention

Your organization might choose to create several different kinds of builds to optimize build times.

- **CI builds:** The purpose of this build is to ensure code compiles and unit tests are run. This build gets triggered at each commit. It serves as the heartbeat of the project and provides quality feedback to the team immediately. For more information, see Specify events that trigger pipelines.
- **Nightly build:** The purpose of a nightly build isn't only to compile the code, but also to ensure any larger test suites that are inefficient run for each build on a regular cadence. Usually, these cadences are integration, UI, or smoke tests. For more information, see Configure schedules for pipelines.
- **Release build:** In addition to compiling and running tests, this build also compiles the API documentation, compliance reports, code signing, and other steps that aren't required every time the code is built. This build provides the golden copy that's pushed to the release pipeline to finally deploy in the production environment.

The types of builds needed by your organization depend on factors including your team's and organization's maturity, the kind of product you're working on, and your deployment strategy.

Next steps

Release Engineering: Deployment

Release engineering: Deployment

Article • 05/31/2023

As you provision and update Azure resources, application code, and configuration settings, a repeatable and predictable process will help you avoid errors and downtime. We recommend automated processes for deployment that you can run on demand and rerun if something fails. After your deployment processes are running smoothly, process documentation can keep them that way.

Automation

To activate resources on demand, deploy solutions rapidly, minimize human error, and produce consistent and repeatable results, be sure to automate deployments and updates.

Automate as many processes as possible

The most reliable deployment processes are automated and idempotent, as in, repeatable to produce the same results.

- To automate your infrastructure, you can use Azure Bicep.
- To configure virtual machines (VMs), you can use cloud-init (for Linux VMs) or Azure Automation State Configuration (DSC).
- To automate application deployment, you can use Azure DevOps Services, GitHub, Jenkins, or other CI/CD solutions.

As a best practice, create a repository of categorized automation scripts for quick access, documented with explanations of parameters and examples of script use. Keep this documentation in sync with your Azure deployments, and designate a primary person to manage the repository.

Automation scripts can also activate resources on demand for disaster recovery. But scripts can sometimes be a place where you accidentally expose secrets. To protect against exposing secrets in your scripts, use secret variables and Azure Key vault. For more information, see Set secret variables and Azure Key Vault.

Automate and test deployment and maintenance tasks

Distributed applications consist of multiple parts that must work together. Deployment should take advantage of proven mechanisms, such as scripts, that can update and validate configuration and automate the deployment process. Test all processes fully to ensure that errors don't cause extra downtime.

Implement deployment security measures

All deployment tools must incorporate security restrictions to protect the deployed application. Define and enforce deployment policies carefully, and minimize the need for human intervention.

Release process

One of the challenges with automating deployment is the cut-over itself, taking software from the final stage of testing to live production. You usually need to do this process quickly in order to minimize downtime. The blue-green deployment approach minimizes downtime by ensuring you have two production environments that are as identical as possible.

Document release process

Without detailed release process documentation, an operator might deploy a bad update or might improperly configure settings for your application. Clearly define and document your release process, and ensure that it's available to the entire operations team.

Stage your workloads

When you deploy to various stages and run tests and validations at each stage before moving on to the next, you ensure friction-free production deployment.

With good use of staging and production environments, you can push updates to the production environment in a highly controlled way and minimize disruption from unanticipated deployment issues.

- *Blue-green deployment* involves deploying an update into a production environment that's separate from the live application. After you validate the deployment, switch the traffic routing to the updated version. One way to switch traffic routing is to use the staging slots available in Azure App Service to stage a deployment before moving it to production.
- *Canary releases* are similar to blue-green deployments. Instead of switching all traffic to the updated application, you route only a small portion of the traffic to

the new deployment. If there's a problem, revert to the old deployment. If not, gradually route more traffic to the new version. In Azure App Service, you can use the testing in production feature to manage a canary release.

Test environments

If development and test environments don't match the production environment, it's hard to test and diagnose problems. So, keep development and test environments as close to the production environment as possible. Make sure that test data is consistent with the data used in production, even if it's sample data and not real production data for privacy or compliance reasons. Plan to generate and anonymize sample test data.

Log and audit

To capture as much version-specific information as possible, implement a robust logging strategy. If you use staged deployment techniques, more than one version of your application is running in production. If a problem occurs, determine which version is causing it.

High availability considerations

An application that depends on a single instance of a service creates a single point of failure. To improve resiliency and scalability, provision multiple instances.

The following are examples of how to provision multiple instances with Azure resources. We recommend that you review the resiliency guidance for the resources that comprise your workload to achieve high availability across it.

- For Azure App Service, select an app service plan that offers multiple instances.
- For Azure Virtual Machines, ensure that your architecture includes more than one VM and that each VM is included in an availability set.

Consider deploying across multiple regions

We recommend deploying all but the least critical applications and application services across multiple regions. If your application is deployed to a single region, in the rare event that the entire region becomes unavailable, the application will also be unavailable. If you choose to deploy to a single region, consider preparing to redeploy to a secondary region as a response to an unexpected failure.

Redeploy to a secondary region

If you run applications and databases in a single, primary region with no replication, your recovery strategy might be to redeploy to another region. This solution is affordable but most appropriate for noncritical applications that can tolerate longer recovery times. If you choose this strategy, automate the redeployment process as much as possible. Also, include redeployment and data synchronization scenarios in your disaster response and testing scenarios.

To automate your redeployment process, consider using Azure Site Recovery.

Next steps

Release engineering: Rollback

Release engineering: Rollback

Article • 05/26/2023

In some cases, a new software deployment can harm or degrade the functionality of a software system. When building your solutions, it's essential to anticipate deployment issues and to architect solutions that provide mechanisms for fixing problematic deployments. Rolling back a deployment involves reverting the deployment to a known good state.

Rollback can be accomplished in many different ways. Several Azure services support native mechanisms for rolling back to a previous state. We recommend you use them where available and where it makes sense for your chosen deployment model.

The following examples help you design your rollback and recovery plan.

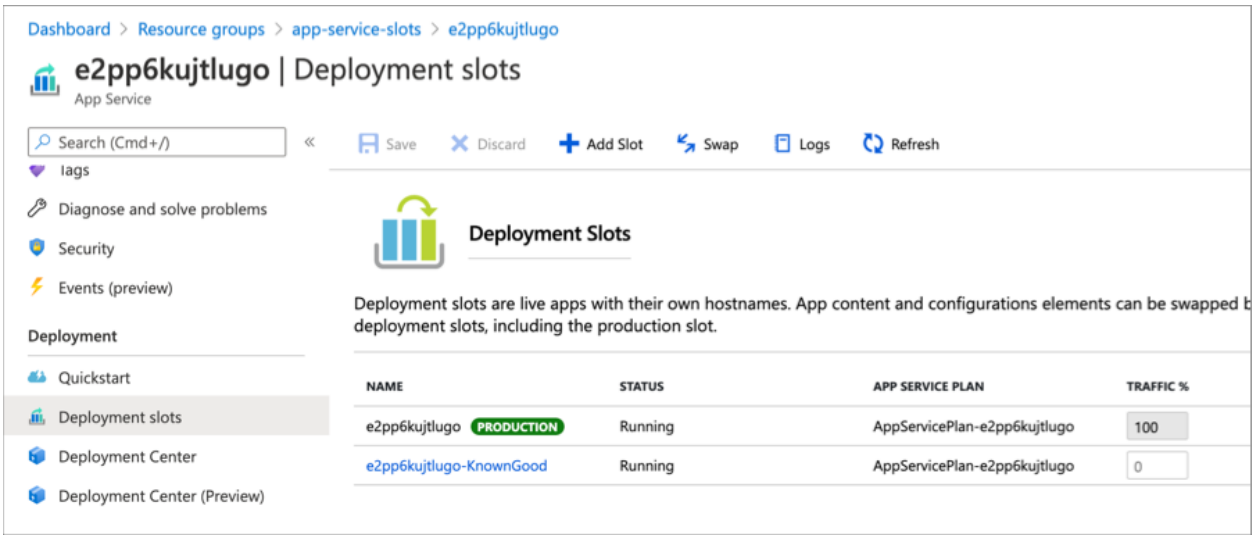
Make use of staged deployments

If you use a staged deployment strategy, you deploy several versions of your application and use networking techniques to switch between them. You can then quickly revert to a previous version of the application if you detect a problem along the way.

With Azure App Service, you can use deployment slots to implement staged deployment. Deployment slots are running instances of the application, each with a separate host name.

Slots can be used to stage and test applications before promoting to a production slot. A deployment slot can be created to hold the last known good instance of your application. If there's an issue or problematic deployment, the production slot can be swapped with the known good slot to bring the application back to a known good state quickly.

The following image shows how to do it by way of an Azure DevOps pipeline.



Learn more

For more information on Azure App Service deployment slots, see Set up staging environments in Azure App Service.

Apply desired state configuration

Depending on the resources in your workload, you might be able to use desired state configurations to automate and govern your deployment process.

Kubernetes is among the solutions that let you express your deployment instructions as a desired state configuration. With Kubernetes, you can use a deployment instruction to describe the particular workload running in the cluster. A deployment might declare that a workload consists of three replicas of a specific pod that should always be running. The deployment object creates a ReplicaSet and the associate Pods. When you update a workload, the deployment itself can be revised, which will generally roll out a new container image to the deployment pods. Assuming multiple replicas of the pods exist, this rollout can happen in a controlled and staged manner and in harmony with your workload's overall deployment strategy.

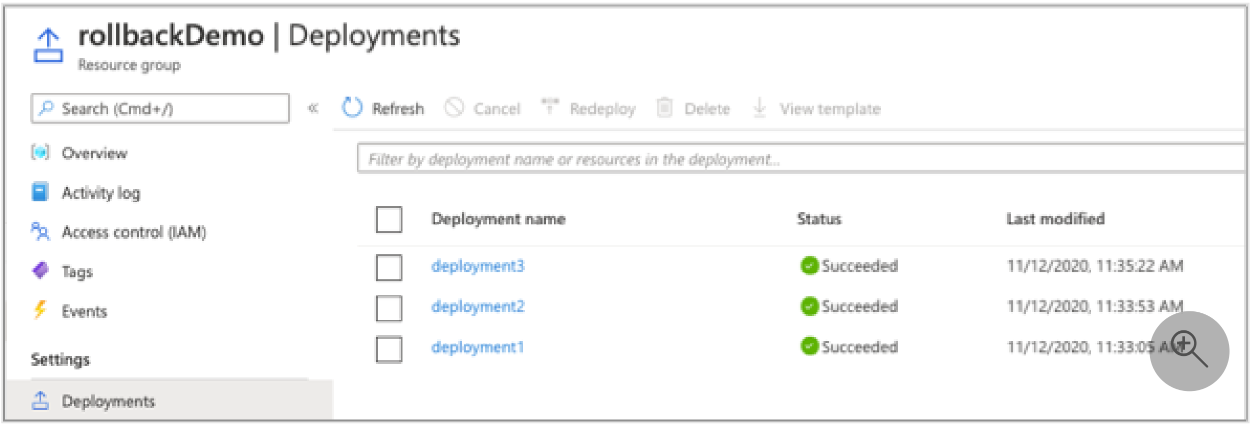
Learn more

For more information, see Kubernetes deployments.

Work with deployment records

A deployment record is created when deploying Azure infrastructure and solutions with Azure Resource Manager (ARM) templates or other infrastructure as code solutions.

When you create a new deployment, you can provide a previously known good deployment so that if the current deployment fails, the previous good deployment is redeployed. There are several considerations and caveats when you use this functionality. For more information, see Rollback on an error to successful deployment.



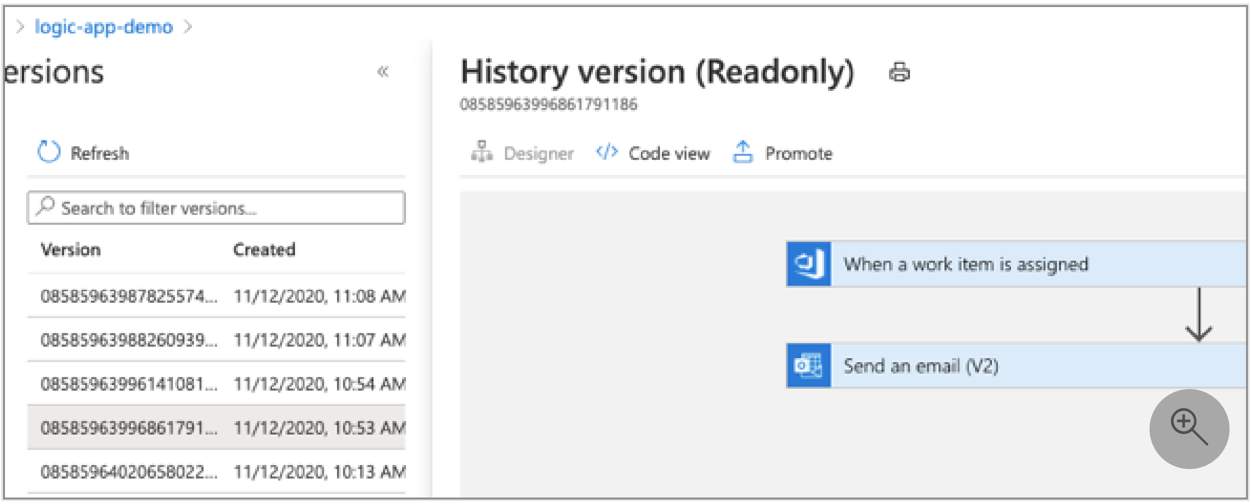
Learn more

For more information, see Rollback on an error to successful deployment.

Restore a known good state

Many managed services have a concept of versioning with a built-in restore function.

Azure Logic Apps, for example, create a new version of the application whenever an update is made to it. Azure maintains a history of versions and can revert or promote any previous version. To do so, in the Azure portal, select your logic app and choose **Versions**. Previous versions can be selected on the versions pane, and the application can be inspected both in the code view and the visual designer view. Select the version you want to revert to, select the **Promote** option, and then **Save**.



Learn more

For more information, see Manage logic apps in the Azure portal.

Restore a database

You can do a point in time restore on any primary database on the same server. You restore to an earlier point in time within the database's retention period.

Learn more

For more information, see Point in time restore

Monitoring operations of cloud applications

Article • 03/15/2023

Distributed applications and services running in the cloud are, by nature, complex pieces of software that include many moving parts. In a production environment, it's important to track the way customers use your system and monitor the health, and performance of your system. Use the following checklist as a diagnostic aid to detect, correct, and prevent issues from occurring.

Checklist

How are you monitoring your workload?

- ✓ Ensure that the system remains healthy.
- ✓ Track the availability of the system and its component elements.
- ✓ Maintain performance to ensure that the throughput of the system doesn't degrade unexpectedly as the volume of work increases.
- ✓ Guarantee that the system meets any service-level agreements (SLAs) established with customers.
- ✓ Protect the privacy and security of the system, users, and their data.
- ✓ Track the operations performed for auditing or regulatory purposes.
- ✓ Monitor the day-to-day usage of the system and spot trends that might lead to problems if they're not addressed.
- ✓ Track issues that occur, from initial report through to analysis of possible causes, rectification, consequent software updates, and deployment.
- ✓ Trace operations and debug software releases.

In this section

Follow these questions to assess the workload at a deeper level.

Assessment	Description
Do you monitor your resources?	Have an overall view of the workload resources. The information can come from application code, frameworks, external sources with which the application communicates, and the underlying infrastructure.

Assessment	Description
Do you have detailed instrumentation in the application code?	Instrumentation lets you gather performance data, diagnose problems, and make decisions.
Do you correlate application events across all application components?	Collect data from various sources, consolidate and clean various formats, and store in reliable storage.
Do you interpret the collected data to spot issues and trends in application health?	Analyze the data collected from various data sources to assess the overall well-being of the workload.
Do you visualize monitoring data?	Present the analyzed data in a way that an operator can quickly spot any trends or problems.
Do you have alerts and response plans ready for the relevant teams when issues occur?	Present the analyzed data in a way that an operator can quickly spot any trends or problems.
Do you use the Azure platform notifications and updates?	Consider the underlying infrastructure such as virtual machines, networks, and storage services to collect important platform-level diagnostic data.
Do you monitor and measure application health?	Health monitoring generates a snapshot of the current health of the system so that you can verify all components are functioning as expected.
Do you monitor and track resource usage?	Usage monitoring tracks how the features and components of an application are used.
Do you collect data from the reported issues	Analyzing data, for unexpected events, can provide insight about the application health.
Do you collect audit logs for regulatory requirements?	Auditing can provide evidence useful for compliance attestations.

Azure offering

You can use any monitoring platform to manage your Azure resources. Microsoft's first party offering is Azure Monitor, a comprehensive solution for metrics and logs from the infrastructure to the application code, including the ability to trigger alerts and automated actions as well as data visualization.

Reference architecture

Enterprise monitoring with Azure Monitor illustrates an architecture that has centralized monitoring management by using Azure Monitor features.

Related links

- Distributed tracing
- Application Insights

Next steps

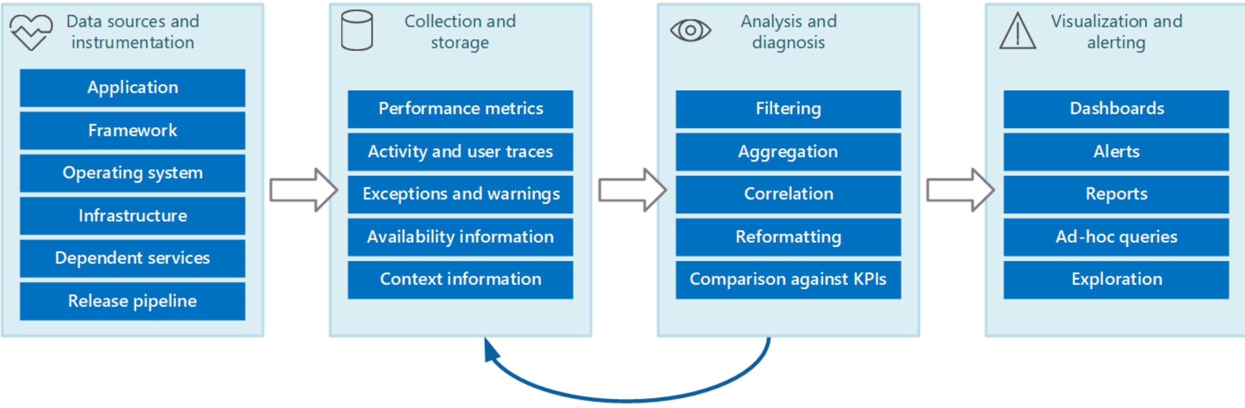
Monitoring phases

Monitoring workloads

Article • 11/30/2022

Monitoring a workload and the infrastructure in which it runs, should not be processed or analyzed in isolation. Build a pipeline that gives you holistic observability of the system.

This article describes the stages of a common pipeline design.



1. Data sources and instrumentation

Monitoring data from many sources:

- **Application code** Developers add trace messages in application code to track the flow of control. For example, recording the entry and exit times can be useful. An entry to a method in the application can emit a trace message that specifies the name of the method, the current time, the value of each parameter, and any other pertinent information.

- **Application frameworks** Many applications use libraries and frameworks to perform common tasks such as accessing a data store or communicating over a network. The frameworks emit trace messages and raw diagnostic information, such as transaction rates and data transmission successes and failures can be important information for debugging issues.

- **Dependent services** The application will access external services such as a web server or database management system to complete business operations. Even a single business operation can result in multiple point-to-point calls among all services. The services might publish their own trace information, logs, and performance counters. Examples include SQL Server Dynamic Management Views for tracking operations performed against a SQL Server database, and IIS trace logs for recording requests made to a web server.

- **Release pipeline** As the components of a system are modified and new versions are deployed, it's important to be able to attribute issues, events, and metrics to each version. This way problems with a specific version of a component can be tracked quickly and rectified.

> ⓘ Note
>
> As a workload owner, you may not be monitoring infrastructure metrics actively. However, this information can indicate systemic issues. Consider the underlying infrastructure and components on which your system runs. Virtual machines, virtual networks, and storage services can all be sources of important infrastructure-level performance counters and other diagnostic data.
>
> Another important source is the operating system where the application runs. It can be a source of low-level system-wide information, such as performance counters that indicate I/O rates, memory utilization, and CPU usage. Operating system errors (such as the failure to open a file correctly) might also be reported.

More information: Sources of monitoring data

Instrumentation

Instrumentation is a process of generating diagnostics data in the data sources. The captured data can help you assess performance, diagnose problems, and make decisions. In the simplest case of a single application, it can be a log of trace messages.

In complex scenarios, there might be a need to *correlate instrumentation data* from several sources with sufficient context so that the data can be mapped it to an action. For example, at the framework level, a task might be identified by a thread ID. Within an application, the same work might be associated with the user ID for the user who is performing that task.

More information: Instrumentation best practices

2. Collection and storage

At this stage, the information generated by instrumentation is collected and transformed so that it's easy to consume and then saved to a reliable storage. The choices for the data collection and storage technologies will depend on some factors:

- Do you need to analyze data quickly (hot path) or it can be stored for later analysis (warm, cold path)?
- Do you want the data collection tier to retrieve data actively (pull model) or wait for the data to arrive (push model)?
- Do you want to consolidate data from various sources before writing to storage?
- Do you need to archive the data?

More information: Data collection and storage considerations

3. Analysis and diagnosis

The analysis and diagnosis stage takes the collected raw data and produces information that can be used to determine the state of the system. This information is useful in deciding remediation actions. In certain cases, the results can be fed back into the instrumentation and collection stages.

More information: Analyzing data

4. Visualization and alerting

The purpose of visualization is to present information in near real time by using a series of dashboards. Also, you can view historical data through reports, graphs, and charts to identify long-term trends.

More information: Visualizing data

Alerting

If information indicates that the key indicators are likely to exceed acceptable bounds, you can trigger alerts to raise awareness. It can trigger an automated process that attempts to take corrective actions, such as autoscaling.

More information: Alerting

Next steps

More information: Sources of monitoring data

Data sources for diagnostic data

Article • 11/30/2022

The information that the monitoring process uses can come from several sources: application code, frameworks, external sources with which the application communicates, and the underlying infrastructure.

This article highlights some best practices of those sources and provides guidance on the types of information that you should capture.

Application code

Developers add trace messages in the application code to track the flow of control. Here are some best practices:

- **Follow a standard approach** For example, recording the entry and exit times can be useful. An entry to a method in the application can emit a trace message that specifies the name of the method, the current time, the value of each parameter, and any other pertinent information.
- **Log all exceptions and warnings** Retain a full trace of any nested exceptions and warnings. Ideally, you should also capture information that identifies the user who is running the code, together with activity correlation information (to track requests as they pass through the system).
- **Log attempts to access all dependent resources** An application will communicate with services such as message queues, databases, files, and other dependent services. This information can be used for metering and auditing purposes.
- **Capture information that identifies the user who is running the code** This information with activity correlation information is useful in tracking requests as they pass through the system.

Application frameworks

Many applications use libraries and frameworks to perform common tasks such as accessing a data store or communicating over a network. Consider configuring the frameworks to emit trace messages. These frameworks might be configurable to provide their own trace messages and raw diagnostic information, such as transaction rates and data transmission successes and failures.

ⓘ Note

Many modern frameworks automatically publish performance and trace events. Capturing this information is simply a matter of providing a means to retrieve and store it where it can be processed and analyzed.

Dependent services

The application will access external services such as a web server or database management system to complete business operations. Even a single business operation can result in multiple point-to-point calls among all services. The services might publish their own trace information, logs, and performance counters. Examples include SQL Server Dynamic Management Views for tracking operations performed against a SQL Server database, and IIS trace logs for recording requests made to a web server.

Operating system

Another important source is the operating system where the application runs. It can be a source of low-level system-wide information, such as performance counters that indicate I/O rates, memory utilization, and CPU usage. Operating system errors such as the failure to open a file correctly might also be reported.

Infrastructure metrics

ⓘ Note

As a workload owner, you may not be monitoring infrastructure metrics actively. However, this information can indicate systemic issues. Consider the underlying infrastructure and components on which your system runs. Virtual machines, virtual networks, and storage services can all be sources of important infrastructure-level performance counters and other diagnostic data.

- **Compute monitoring** Collect metrics from compute resources on which the application is running. This might be virtual machines, App Services, or Kubernetes. Knowing the state of your infrastructure will allow you to react promptly if there are any issues.
- **Data tier monitoring** Include metrics the databases, storage accounts, and other data sources that interact with the application. A low performance of the data tier

of an application could have serious consequences.

- **Container monitoring** If your application run on Azure Kubernetes Service (AKS), you'll need to monitor the state of your cluster, nodes, and pods. One option is to the container insights feature in Azure Monitor. This feature delivers quick, visual, and actionable information: from the CPU and memory pressure of your nodes to the logs of individual Kubernetes pods.

  Operators who prefer using the open-source Kubernetes monitoring tool Prometheus can take advantage of its integration with container insights.

  The Sidecar Pattern adds a separate container with responsibilities that are required by the main container. A common use case is for running logging utilities and monitoring agents.

- **Network monitoring** Networking is key to an application running without issues. Consider using Network Watcher, a collection of network monitoring and troubleshooting tools. Some of these tools are:
  - Traffic Analytics shows the flows the virtual networks and uses Microsoft Threat Intelligence databases to give you percentage traffic from malicious IP addresses. You can identify bottlenecks by seeing the systems in your virtual networks that generate most traffic.
  - Network Performance Monitor can generate synthetic traffic to measure the performance of network connections over multiple links, giving you a perspective on the evolution of WAN and Internet connections over time, as well as offering valuable monitoring information about Microsoft ExpressRoute circuits.
  - VPN diagnostics can help troubleshooting site-to-site VPN connections connecting your applications to users on-premises.

Release pipeline

As the components of a system are modified and new versions are deployed, it's important to be able to attribute issues, events, and metrics to each version.

- **Relate information about issues, events, and others to the release pipeline** Problems with a specific version of a component can be tracked quickly and rectified.

- **Log security-related information for successful and failing requests** Security issues might occur at any point in the system. For example, a user might attempt to sign in with an invalid user ID or password. An authenticated user might try to obtain unauthorized access to a resource. Or a user might provide an invalid or outdated key to access encrypted information.

Sources from application and system monitoring

This strategy uses internal sources within the application, application frameworks, operating system, and infrastructure. The application code can generate its own monitoring data at notable points during the lifecycle of a client request. The application can include tracing statements that might be selectively enabled or disabled as circumstances dictate. It might also be possible to inject diagnostics dynamically by using a diagnostics framework. These frameworks typically provide plug-ins that can attach to various instrumentation points in your code and capture trace data at these points.

Additionally, your code and/or the underlying infrastructure might raise events at critical points. Monitoring agents that are configured to listen for these events can record the event information.

Real user monitoring

This approach records the interactions between a user and the application and observes the flow of each request and response. This information can have a two-fold purpose: it can be used for metering usage by each user, and it can be used to determine whether users are receiving a suitable quality of service (for example, fast response times, low latency, and minimal errors). You can use the captured data to identify areas of concern where failures occur most often. You can also use the data to identify elements where the system slows down, possibly due to hotspots in the application or some other form of bottleneck. If you implement this approach carefully, it might be possible to reconstruct users' flows through the application for debugging and testing purposes.

> ⓘ **Important**
>
> You should consider the data that's captured by monitoring real users to be highly sensitive because it might include confidential material. If you save captured data, store it securely. If you want to use the data for performance monitoring or debugging purposes, strip out all personally identifiable information first.

Synthetic user monitoring

In this approach, you write your own test client that simulates a user and performs a configurable but typical series of operations. You can track the performance of the test client to help determine the state of the system. You can also use multiple instances of the test client as part of a load-testing operation to establish how the system responds under stress, and what sort of monitoring output is generated under these conditions.

> ⓘ **Note**
>
> You can implement real and synthetic user monitoring by including code that traces and times the execution of method calls and other critical parts of an application.

Application profiling

This approach is primarily targeted at monitoring and improving application performance. Rather than operating at the functional level of real and synthetic user monitoring, it captures lower-level information as the application runs. You can implement profiling by using periodic sampling of the execution state of an application (determining which piece of code that the application is running at a given point in time). You can also use instrumentation that inserts probes into the code at important junctures (such as the start and end of a method call) and records which methods were invoked, at what time, and how long each call took. You can then analyze this data to determine which parts of the application might cause performance problems.

Endpoint monitoring

This technique uses one or more diagnostic endpoints that the application exposes specifically to enable monitoring. An endpoint provides a pathway into the application code and can return information about the health of the system. Different endpoints can focus on various aspects of the functionality. You can write your own diagnostics client that sends periodic requests to these endpoints and assimilate the responses. For more information, see the Health Endpoint Monitoring pattern.

For maximum coverage, you should use a combination of these techniques.

Next steps

Instrumentation

Instrument an application

Article • 11/30/2022

Instrumentation is a critical part of the monitoring process. You can make meaningful decisions about the performance and health of a system only if you first capture the data that enables you to make these decisions. By using instrumentation, the information that you gather should be sufficient to assess performance, diagnose problems, and make decisions without requiring you to sign in to a remote production server to perform tracing, and debugging manually. Instrumentation data typically includes metrics and information that's written to trace logs.

Trace logs

The contents of a trace log can be the result of textual data that's written by the application or binary data that's created as the result of a trace event, if the application is using Event Tracing for Windows (ETW). Trace log contents can also be generated from system logs that record events arising from parts of the infrastructure, such as a web server. Textual log messages are often designed to be human-readable, but they should also be written in a format that enables an automated system to parse them easily.

Categorize logs and use separate logs to record the trace output from different operational aspects of the system, instead of writing all trace data to a single log. You can then quickly filter log messages by reading from the appropriate log rather than having to process a single lengthy file. Never write information that has different security requirements (such as audit information and debugging data) to the same log.

> ⓘ **Note**
>
> A log may be implemented as a file on the file system, or it may be held in some other format, such as a blob in blob storage. Log information may also be held in more structured storage, such as rows in a table.

Metrics

Metrics will generally be a measure or count of some aspect or resource in the system at a specific time, with one or more associated tags or dimensions, sometimes called a *sample*. A single instance of a metric isn't useful in isolation. Instead, metrics have to be captured over time. The key issue to consider is which metrics you should record and how frequently. Generating data for metrics too often can impose a significant extra load on the system; whereas, capturing metrics infrequently may cause you to miss the circumstances that lead to a significant event. The considerations will vary from metric to metric. For example, CPU utilization on a server might vary significantly from second to second, but high utilization becomes an issue only if it's long-lived over many minutes.

Information for correlating data

You can easily monitor individual system-level performance counters, capture metrics for resources, and obtain application trace information from various log files. Some forms of monitoring require the analysis and diagnostics stage in the monitoring pipeline to correlate the data that's retrieved from several sources. This data may take several forms in the raw data, and the analysis process must be provided with sufficient instrumentation data to map these different forms. For example, at the application framework level, a task may be identified by a thread ID. Within an application, the same work may be associated with the user ID for the user who is completing that task.

Also, it's unlikely to be a 1:1 mapping between threads and user requests, because asynchronous operations may reuse the same threads to do operations for more than one user. To complicate matters further, a single request may be handled by more than one thread as execution flows through the system. If possible, associate each request with a unique activity Id that's propagated through the system as part of the request context. The technique for generating and including activity IDs in trace information depends on the technology that's used to capture the trace data.

All monitoring data should be timestamped in the same way. For consistency, record all dates and times by using Coordinated Universal Time, which helps you trace sequences of events with ease.

> ⓘ **Note**
>
> Computers operating in different time zones and networks may not be synchronized. Don't depend on timestamps alone for correlating instrumentation data that spans multiple machines.

Information to include in the instrumentation data

Consider the following points when you're deciding which instrumentation data you need to collect.

Human-readable data

Ensure that information captured by trace events is machine and human readable. Adopt well-defined schemas for this information to help automated processing of log data across systems, and to provide consistency to operations and engineering staff reading the logs.

Include the following environmental information, such as:

- Deployment environment
- Machine on which the process is running
- Details of the process
- Call stack

Consider performance tradeoffs

Enable profiling only when necessary because it can impose a significant overhead on the system. By using instrumentation, profiling records an event, such as a method call, every time it occurs; but, sampling records only selected events.

The selection can be time-based, such as once every `n` seconds, or frequency-based, such as once every `n` requests. If events occur frequently, profiling by instrumentation may cause too much of a burden and effect overall performance. In this case, the sampling approach may be preferable.

However, if the frequency of events is low, sampling may miss them. In this case, instrumentation may be the better approach.

Invest in traceability and correlation

Provide sufficient context to enable a developer or administrator to determine the source of each request, which may include some form of activity Id that identifies a specific instance of a request.

It may also include information that can be used to correlate this activity with the computational work performed and the resources used. *This work might cross process and machine boundaries.*

For metering, the context should also include, either directly or indirectly, through other correlated information, a reference to the customer who caused the request. This context provides valuable information about the application state at the time that the monitoring data was captured.

Capture all relevant data

Record all requests, and the locations, or regions from which these requests are made. This information can help determine whether there are any location-specific hotspots. This information can also be useful to determine whether to repartition an application or the data that it uses.

Record and capture the details of exceptions carefully. Often, critical debug information is lost because of poor exception handling.

Capture the full details of exceptions that the application throws, including any inner exceptions and other context information. Include the call stack, if possible.

Strive for data consistency

Consistent data can help analyze events and correlate them with user requests. Consider using a comprehensive and configurable logging package to gather information, rather than depending on developers to adopt the same approach as they implement different parts of the system.

Gather data from key performance counters, such as the volume of I/O being performed, network utilization, number of requests, memory use, and CPU utilization.

Some infrastructure services may provide their own specific performance counters, such as:

- The number of connections to a database.
- The rate at which transactions are performed.
- The number of transactions that succeed or fail. Applications might also define their own specific performance counters.

Consider external dependencies

Log all external service calls made.

For example to:

- Database systems

- Web services
- Other system-level services that are part of the infrastructure

Record information about the time taken to perform each call, and the success or failure of the call. If possible, capture information about all retry attempts and failures for any transient errors that occur.

Ensure compatibility with telemetry systems

In many cases, the instrumentation information is generated as a series of events and passed to a separate telemetry system for processing and analysis. A telemetry system is typically independent of any specific application or technology, but it expects information to follow a specific format that's defined by a schema. The schema effectively specifies a contract that defines the data fields and types that the telemetry system can ingest. The schema should be generalized to allow for data arriving from a range of platforms and devices.

A common schema should include fields that are common to all instrumentation events, such as:

- Event name
- Event time
- IP address of the sender
- Details required for event correlation, such as:
  - User ID
  - Device ID
  - Application ID

Remember that any number of devices may raise events, so the schema shouldn't depend on the device type. Various devices may raise events for the same application. The application may support roaming or some other form of cross-device distribution.

The schema may also include domain fields that are relevant to a particular scenario that's common across different applications, such as:

- Information about exceptions
- Application start and end events
- Success or failure of web service API calls

All applications that use the same set of domain fields should emit the same set of events, enabling a set of common reports and analytics to be built.

Finally, a schema may contain custom fields for capturing the details of application-specific events.

Best practices for instrumenting applications

The following list summarizes best practices for instrumenting a distributed application running in the cloud:

- Make logs easy to read and easy to parse. Use structured logging where possible.
- Be concise and descriptive in log messages.
- Identify the source of the log.
- Add timing information as each log record is written.
- Use the same time zone and format for all timestamps.
- Categorize logs and write messages to the appropriate place.
- Don't reveal sensitive information about the system or personal information about users.
    - Scrub this information before it's logged, but ensure that you keep the relevant details.
- Log all critical exceptions, but enable the administrator to turn logging on and off for lower levels of exceptions and warnings.
- Capture and log all retry logic information. This data can be useful in monitoring the transient health of the system.
- Trace out process calls, such as requests to external web services or databases.
- Don't mix log messages with different security requirements in the same log file.
- Except for auditing events, make sure that all logging calls are *fire-and-forget* operations that don't block the progress of business operations.
    - Auditing events are exceptional because they're critical to the business.
- Make sure that logging is extensible (for example in code through the use of an interface) and doesn't have any direct dependencies on a concrete target.
- Make sure that all logging is fail-safe and never triggers any cascading errors.
- Treat instrumentation as an ongoing iterative process and review logs regularly, not just when there's a problem.

Next steps

Collection and storage

Collect, aggregate, and store monitoring data for cloud applications

Article • 11/30/2022

In a distributed system, instrumentation can generate large volumes of data from many components, held in various locations, and in different formats.

For example, an application code might generate trace log files and application event log data. From the infrastructure perspective, performance counters are captured through other technologies. Also, any third-party components and services that the application uses might provide their own instrumentation information in different formats, by using separate trace files, blob storage, or even a custom data store.

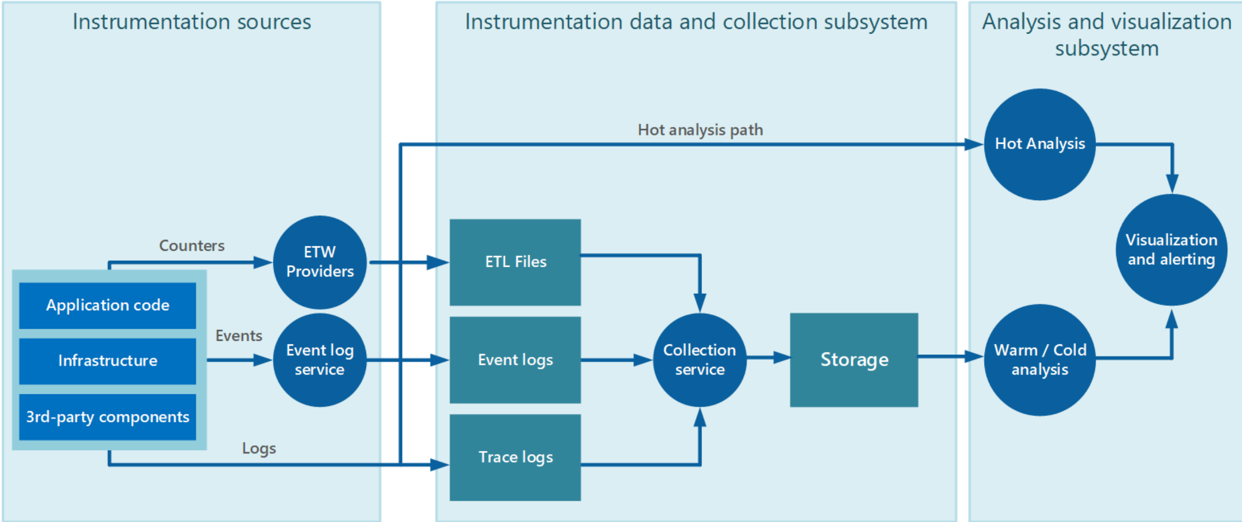
This article provides best practices for collecting data from various sources, consolidating and cleaning various formats, and storing in reliable storage.

Best practices

- Have technologies to collect and aggregate application logs across environments, diagnostics logs from application resources, infrastructure, and other critical dependencies.
- The collection service should run as an out-of-process service and should be simple to deploy.
- All output from the collection service should be a structured and agnostic format.
- The monitoring and data-collection process must be fail-safe and must not trigger any cascading error conditions.
- In the event of a transient failure in sending information to a data sink, the collection service should be prepared to reorder telemetry data so that the newest information is sent first. (The monitoring agent/data-collection service might elect to drop the older data, or save it locally and transmit it later to catch up, at its own discretion.)

Architecture

This image shows a typical instrumentation data-collection process.



- Data collection service collects data from various sources is not necessarily a single process and might comprise many constituent parts running on different machines.
- If you need data to be analyzed quickly, local components that operate outside the collection service might perform the analysis tasks immediately. The results can be sent directly to the visualization and alerting systems.
- If you don't need immediate analysis, data is held in storage while it awaits processing.

The hot, warm, and cold analysis patterns are summarized in Analysis patterns.

Application data

For applications, the collecting service can be an **Application Performance Management (APM) tool** that can run autonomously from the application that generates the instrumentation data. After APM is enabled, you'll have clear visibility of important metrics both in real time and historically. Consider an appropriate level of logging. Verbose logging can incur significant costs.

An example of an APM is Application Insights that aggregates application level logs and events for subsequent analysis.

Application logs support the end-to-end application lifecycle. Logging is essential in understanding how the application operates in various environments and what events occur and under which conditions.

Collecting application logs and events across all major environments is recommended. Separate the data between environments as much as possible. Have filters to ensure non-critical environments do not convolute production log interpretation. Furthermore, corresponding log entries across the application should capture a correlation ID for their respective transactions.

Application events should be captured as a structured data type with machine-readable data points rather than unstructured string types. A structured format, following well-known schema can help in parsing and analyzing logs. Also, structured data can easily be indexed and searched, and reporting can be greatly simplified.

Also the data should be an agnostic format that's independent of the machine, operating system, or network protocol. For example, emit information in a self-describing format such as JSON, MessagePack, or Protobuf rather than ETL/ETW. Using a standard format enables the system to construct processing pipelines; components that read, transform, and send data in the agreed format can be easily integrated.

Infrastructure data

You will also need to collect platform diagnostics to get a holistic view. For example, Windows event logs, performance counters, diagnostic infrastructure logs, and logs from the management plane. Azure platform logs addresses all those needs. Here are some recommendations:

- Collect **Azure Activity Logs** to get audit information. These logs are useful in detecting configuration changes to Azure resources.
- Enable **resource- or infrastructure- level monitoring** throughout the application. This type of logs includes information emitted by platform services such as Azure VMs, Express Route or SQL Database, and also third-party solutions. Configure application resources to route diagnostic logs and metrics to the chosen log aggregation technology.
- **Enforce consistency.** You can use Azure Policy to ensure the consistent use of diagnostic settings across the application, to enforce the desired configuration for each Azure service.
- **Collect logs and metrics available for critical internal dependencies.** This information gives you visibility into the operational state of critical internal dependencies, such as a shared NVA or Express Route connections, and others.

There are many options for a collection service for aggregating infrastructure and resource logs. Azure Log Analytics or Splunk, are popular choices for collating logs and metrics across all application components for subsequent evaluation. Resources may include Azure IaaS, PaaS services, and third-party appliances such as firewalls or anti-malware solutions used in the application. For instance, if Azure Event Hubs is used, the Diagnostic Settings should be configured to push logs and metrics to the data sink.

Understanding usage helps with right-sizing of the workload, but additional cost for logging needs to be accepted and included in the cost model.

Collection strategies

Avoid retrieving telemetry data manually from every component. Have a way of moving the data a central location and consolidated. For a multiregion solution, it's recommended that you first collect, consolidate, and store data on a region-by-region basis, and then aggregate the regional data into a single central system.

To optimize the use of bandwidth, prioritize based on the importance of data. You can transfer less urgent data in batches. However, the data must not be delayed indefinitely, especially if it contains time-sensitive information.

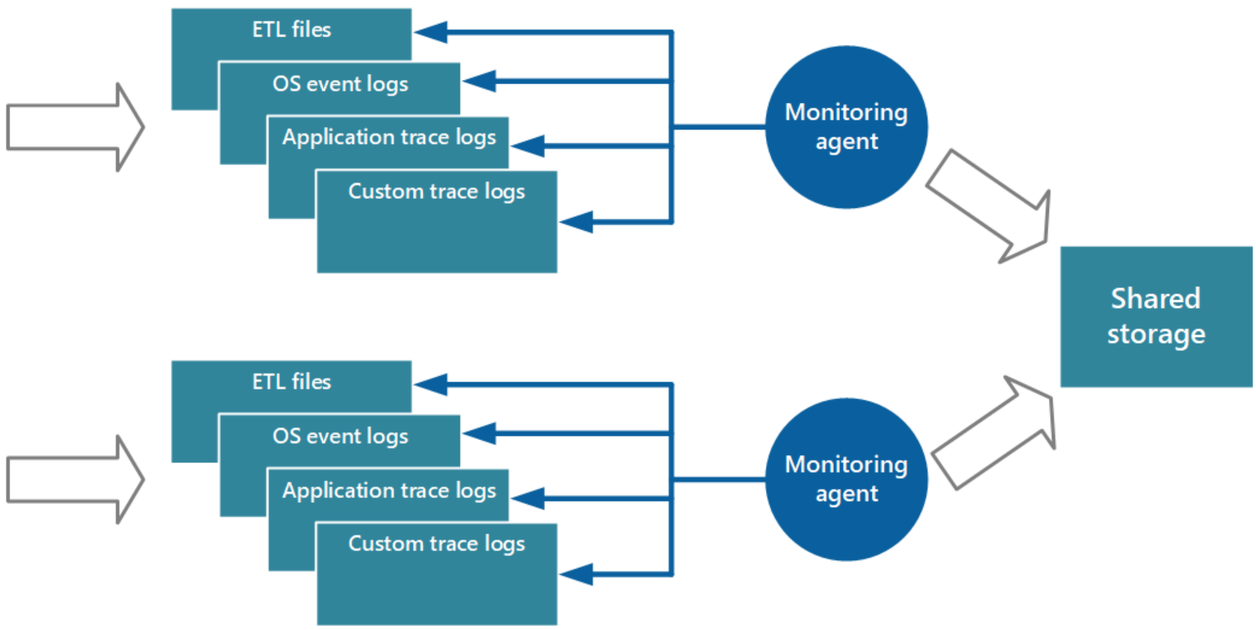
Data collection models

The collection service can collect instrumentation data in mainly two models:

- **Pull model** Actively retrieves data from the various logs and other sources for each instance of the application.
- **Push model** Passively waits for the data to be sent from the components that constitute each instance of the application.

Monitoring agents

Monitoring agents work in pull model. Agents run locally in a separate process with each instance of the application and periodically pull data and write this information directly to centralized storage shared by all instances of the application.



For more information, see Azure Diagnostics extension.

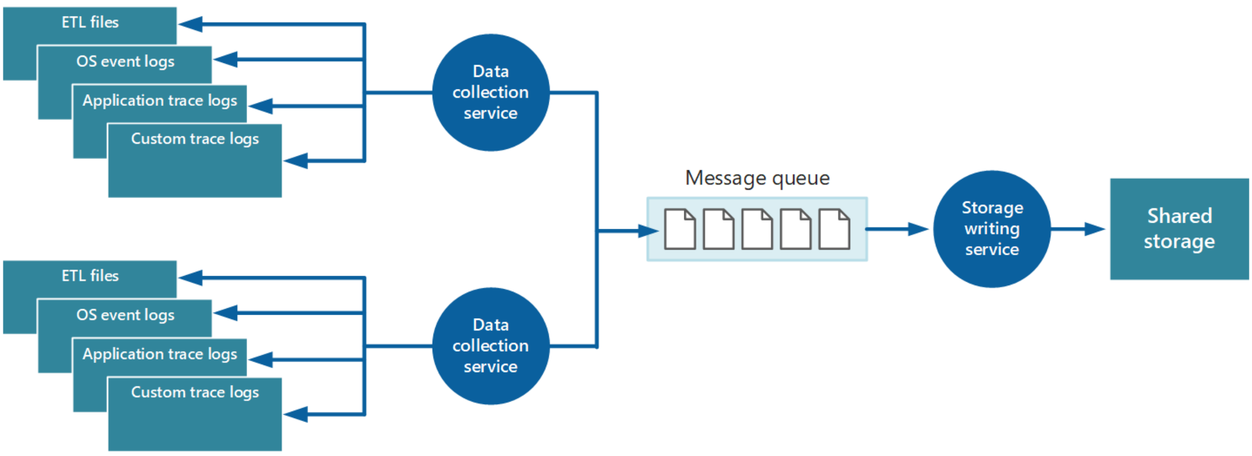
ⓘ Note

Using a monitoring agent is ideally suited to capturing instrumentation data that's naturally pulled from a data source. It's appropriate for a small-scale application running on a limited number of nodes in a single location. An example is information from SQL Server Dynamic Management Views or the length of an Azure Service Bus queue.

Performance considerations

A complex, highly scalable, application might generate huge volumes of data. The amount of data can easily overwhelm the I/O bandwidth available with a single, central location. The telemetry solution must not act as bottleneck and must be scalable as the system expands. Ideally, the solution should incorporate a degree of redundancy to reduce the risks of losing important monitoring information (such as auditing or billing data) if part of the system fails.

One approach is through queuing.

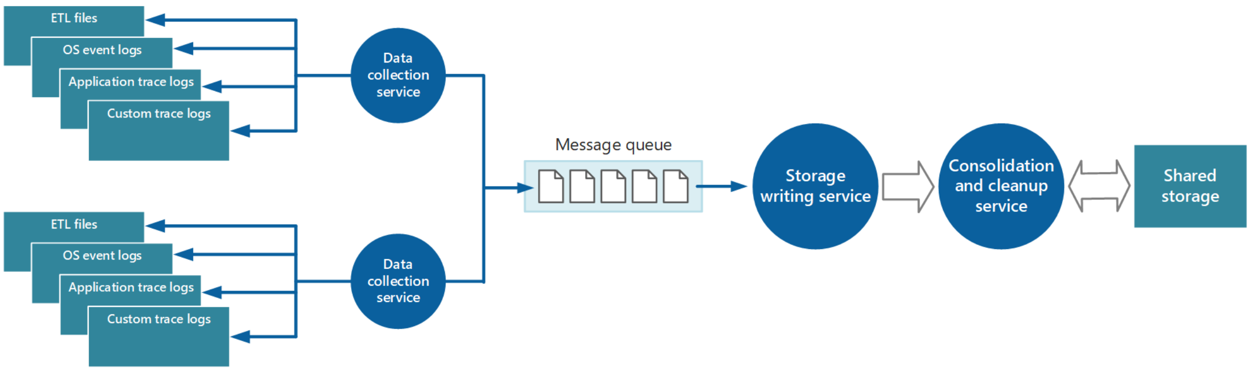


In this architecture, the data-collection service posts data to a queue. A message queue is suitable because it provides "at least once" semantics that help ensure that queued data will not be lost after it's posted. You can implement the storage writing service by using a separate worker role. You can implement this with the Priority Queue pattern.

For scalability, you can run multiple instances of the storage writing service. If there is a high volume of events, you can use Event Hubs to dispatch the data to a different compute for processing and storage.

Consolidation strategies

The data collected from a single instance of an application gives a localized view of the health and performance of that instance. To assess the overall health of the system, it's necessary to consolidate some aspects of the data in the local views. You can perform this after the data has been stored, but in some cases, you can also achieve it as the data is collected.



The instrumentation data can pass through a separate data consolidation service that combines data and acts as a filter and cleanup process. For example, instrumentation data that includes the same correlation information such as an activity ID can be amalgamated. (It's possible that a user starts performing a business operation on one node and then gets transferred to another node if a node fails, or depending on how load balancing is configured.) This process can also detect and remove any duplicated data (always a possibility if the telemetry service uses message queues to push instrumentation data out to storage).

Storage strategies

When deciding the storage capabilities, consider the type of data, how it's used, and how urgently the data is required.

Storage technologies

Consider a polyglot persistence approach where different types of information are stored by using technologies that are most appropriate to the way in which each type is likely to be used.

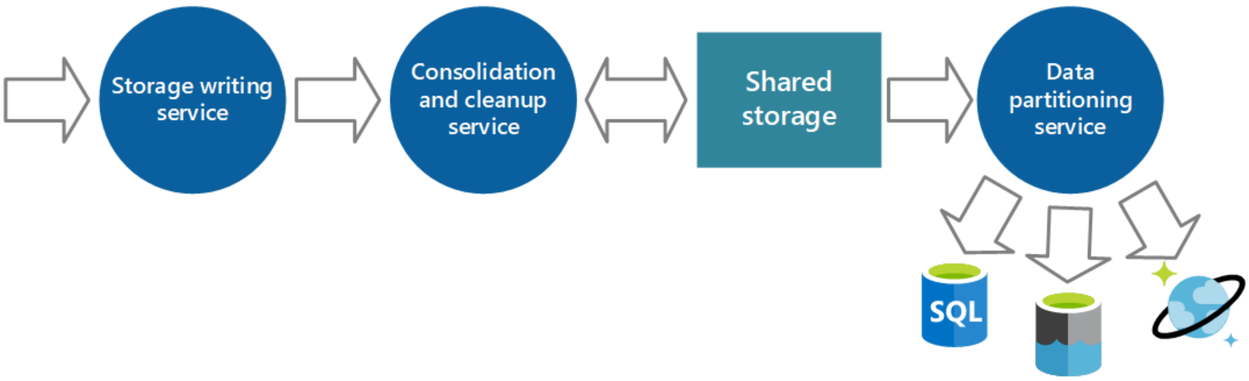
For example, Azure blob and table storage have some similarities in the way in which they're accessed. But they have differences in the operations you can perform for each, and the granularity of the data that they hold. If you need to perform more analytical operations or require full-text search capabilities on the data, it might be more appropriate to use data storage that provides capabilities that are optimized for specific types of queries and data access. For example:

- Performance counter data can be stored in a SQL database to enable ad hoc analysis.
- Trace logs might be better stored in Azure Cosmos DB.
- Security information can be written to HDFS.
- Information that requires full-text search can be stored through Elasticsearch (which can also speed searches by using rich indexing).

The same instrumentation data might be required for more than one purpose. For example, performance counters can be used to provide a historical view of system performance over time. This information might be combined with other usage data to generate customer billing information. In these situations, the same data might be sent to more than one destination, such as a document database that can act as a long-term store for holding billing information, and a multidimensional store for handling complex performance analytics.

Consolidation service

You can implement another service that periodically retrieves the data from shared storage, partitions and filters the data according to its purpose, and then writes it to an appropriate set of data stores.



An alternative approach is to include this functionality in the consolidation and cleanup process and write the data directly to these stores as it's retrieved rather than saving it in an intermediate shared storage area.

Each approach has its advantages and disadvantages. Implementing a separate partitioning service lessens the load on the consolidation and cleanup service, and it enables at least some of the partitioned data to be regenerated if necessary (depending on how much data is retained in shared storage). However, it consumes additional resources. Also, there might be a delay between the receipt of instrumentation data from each application instance and the conversion of this data into actionable information.

Querying considerations

Consider how urgently the data is required. Data that generates alerts must be accessed quickly, so it should be held in fast data storage and indexed or structured to optimize the queries that the alerting system performs. In some cases, it might be necessary for the collection service to format and save data locally so that a local instance of the alerting system can send notifications quickly. The same data can be dispatched to the storage writing service shown in the previous images and stored centrally if it's also required for other purposes.

Data retention considerations

In some cases, after the data has been processed and transferred, the original raw source data that was stored locally can be removed. In other cases, it might be necessary or useful to save the raw information. For example, data that's generated for debugging purposes might be best left available in its raw form but can then be discarded quickly after any bugs have been rectified.

Performance data often has a longer life so that it can be used for spotting performance trends and for capacity planning. The consolidated view of this data is usually kept online for a finite period to enable fast access. After that, it can be archived or discarded.

It's useful to store historical data so you can spot long-term trends. Rather than saving old data in its entirety, it might be possible to down-sample the data to reduce its resolution and save storage costs. As an example, rather than saving minute-by-minute performance indicators, you can consolidate data that's more than a month old to form an hour-by-hour view.

Data gathered for metering and billing customers might need to be saved indefinitely. Additionally, regulatory requirements might dictate that information collected for auditing and security purposes also needs to be archived and saved. This data is also sensitive and might need to be encrypted or otherwise protected to prevent tampering. You should never record users' passwords or other information that might be used to commit identity fraud. Such details should be scrubbed from the data before it's stored.

Next step

Information that's used for more considered analysis, for reporting, and for spotting historical trends is less urgent and can be stored in a manner that supports data mining and ad hoc queries.

Analyze monitoring data

Analyze monitoring data for cloud applications

Article • 04/10/2023

After you've collected data from various data sources, analyze the data to assess the overall well-being of the system. For analysis, have a clear understanding of:

- How to structure data based on KPIs and performance metrics you've defined.
- How to correlate the data captured in different metrics and log files. This is important when tracking a sequence of events and help diagnose problems.

In most cases, data for each part component of the architecture is captured locally and then accurately combined with data generated by other components.

For example, a three-tier application has:

- Presentation tier that allows a user to connect to a website
- Middle tier that hosts a set of microservices that processes business logic
- Database tier that stores data associated with the operation

The usage data for a single business operation might span across all three tiers. This information needs to be correlated to provide an overall view of the resource and processing usage for the operation. The correlation might involve some preprocessing and filtering of data on the database tier. On the middle tier, common tasks are aggregation and formatting.

Best practices

- **Correlate application and resource level logs**—Evaluate data at both levels to optimize the detection of issues and troubleshooting of detected issues. You can aggregate the data in a single data sink or have ways to query events across both levels. Using a unified solution, such as Azure Log Analytics, is recommended to aggregate and query application and resource level logs.

- **Define clear retention times on storage for cold analysis**—This practice is recommended to allow historic analysis over a specific period. Another benefit is control on control storage costs. Have processes that make sure data gets archived to cheaper storage and aggregate data for long-term trend analysis.

- **Analyze long-term trends analyzed to predict operational issues—**Evaluate across long-term data to form operational strategies and also to predict what operational issues are likely to occur and when. For instance, if the average response times have been slowly increasing over time and getting closer to the maximum target.

Correlate data

Data generated by instrumentation and captured by Application Performance Monitoring (APM) tools can provide a snapshot of the system state. The main purpose of the snapshot to make this data actionable.

For example:

- What has caused an intense I/O loading at the system level at a specific time?
- Is it the result of a large number of database operations?
- Is this reflected in the database response times, the number of transactions per second, and application response times at the same juncture?

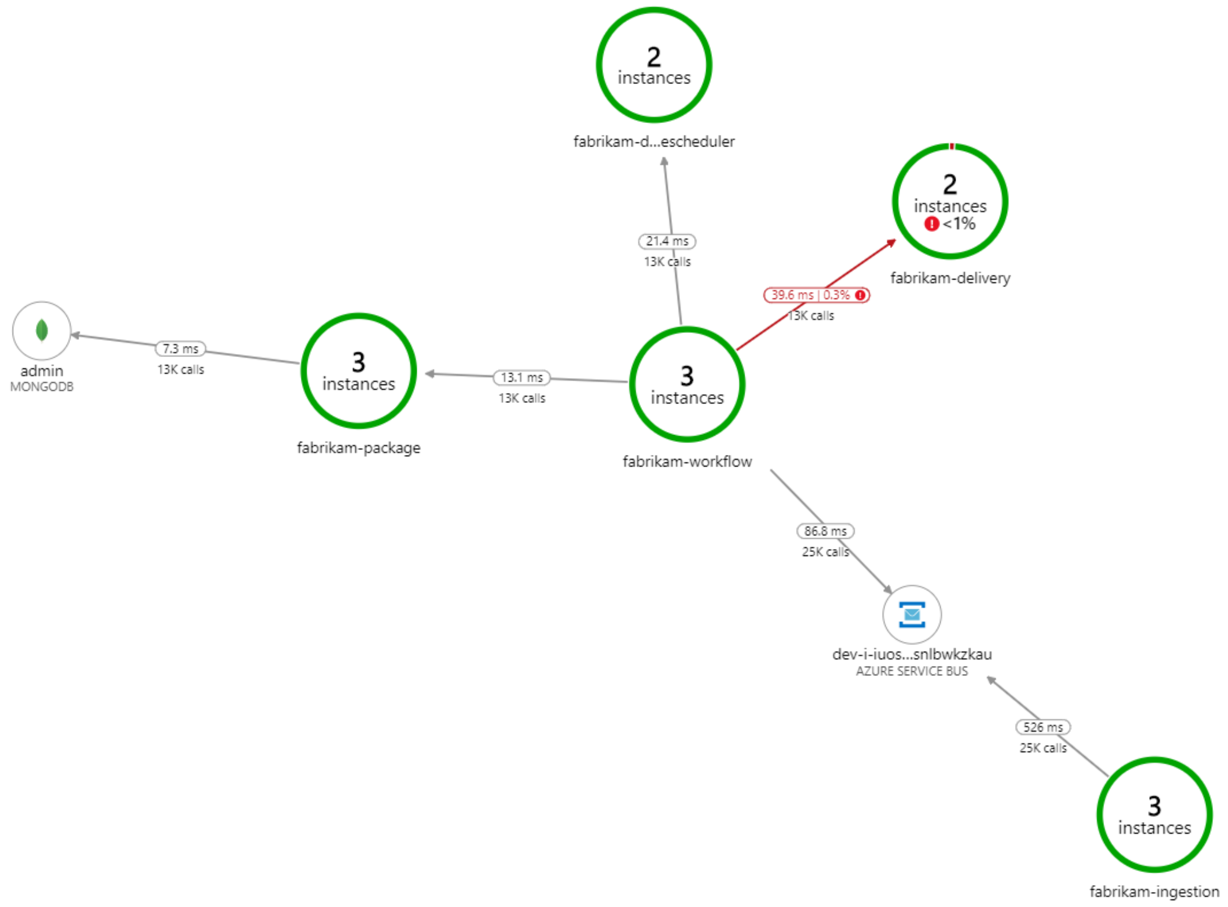
A possible remedial action to reduce the load might be to shard the data over more servers. In addition, exceptions can arise because a fault in any tier. An exception in one level often triggers another fault in the level above.

So it's recommended that you correlate different types of monitoring data at each level to produce an overall view of the state of the system and the applications that are running on it. This is vital for root cause analysis for failures. You can also use this information to make decisions about whether the system is functioning acceptably or not, and determine what can be done to improve the quality of the system.

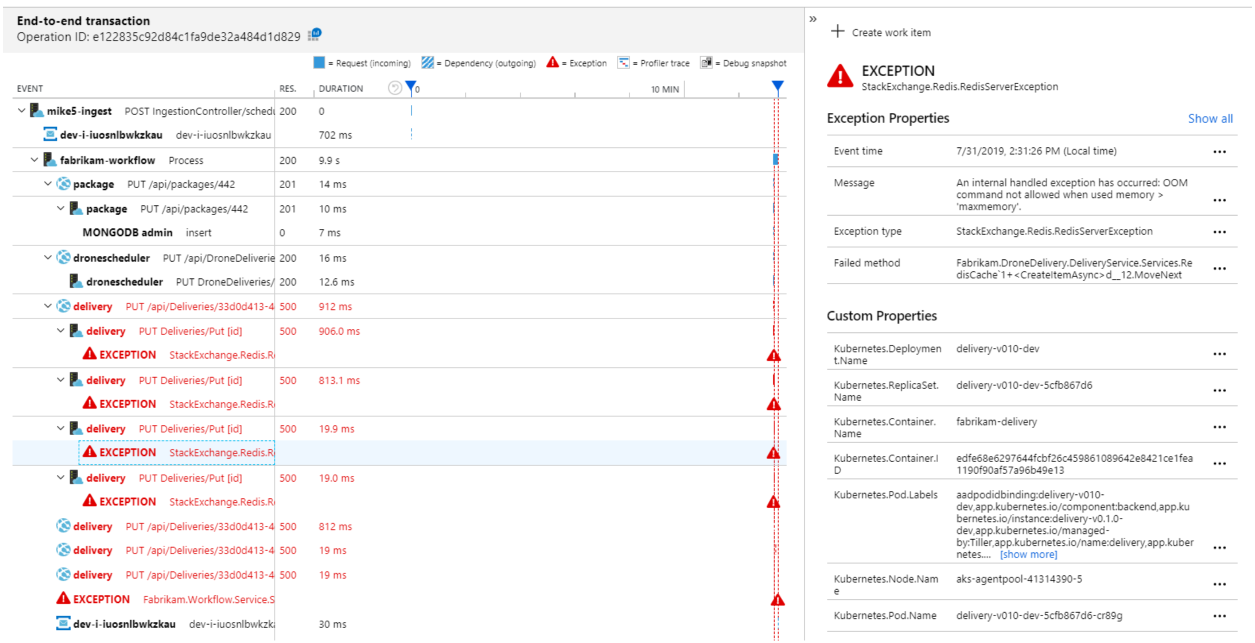
In a distributed application, an operation will lead to multiple transactions. Each transaction generates events in different services and those events must be correlated and visualized. This way you can spot performance bottlenecks or failure hotspots across services. Application Map is a popular choice for visualizing flows.

Telemetry correlation ensures that the transactions can be mapped through the end-to-end application and critical system flows. Platform-level metrics and logs such as CPU percentage, network in/out, and disk operations/sec should be collected from the application to inform a health model and detect/predict issues. This can also help to distinguish between transient and non-transient faults.

Here's an Application Map of an application that has several microservices. With this visual representation, you can see that Workflow service is getting errors from the Delivery service.



When you look at the end-to-end transaction, you can see that an exception is thrown due to memory limits in Azure Cache for Redis.



When correlating data make sure that the raw instrumentation data includes sufficient context and activity ID information to support the required aggregations for correlating events. Additionally, this data might be held in different formats, and it might be necessary to parse this information to convert it into a standardized format for analysis.

Analysis patterns

Analyzing and reformatting data for visualization, reporting, and alerting purposes can be a complex process that consumes its own set of resources. Some forms of monitoring are time-critical and require immediate analysis of data to be effective. While other Other forms of analysis are less time-critical and might require some computation and aggregation after the raw data has been received. This table shows the patterns for analysis.

Pattern	Characteristics	Considerations	Use case
Hot analysis	Time-critical and requires immediate analysis.	• Data must be structured and available quickly for efficient processing. • Move the analysis processing to the individual VMs that stores the data.	• Alerting. • Detecting a security attack on the system.
Warm analysis	Less time-critical and might require aggregation before analysis.	• Perform computation and aggregation on the received raw data. • Aggregate a series of events instead of an isolated event.	• Performance analysis on data from a series of events to identify reliability issues. • Diagnosis of health issues by statistical evaluation of the events leading up to a health event.
Cold analysis	Analysis can be performed at a later date.	• Data is stored safely after it has been captured. • Aggregate a series of events instead of an isolated event.	• Usage monitoring and auditing needs a view of the system at points in time. • Predictive health analysis with historical information.

Combined approach

In common monitoring use cases, you'll use a combination of all three patterns.

For example, a health event is typically processed through hot analysis and can raise an alert immediately. An operator can then drill into the reasons for the health event by examining the data from the warm path. This data should contain information about the events leading up to the issue that caused the health event. An operator can also use cold analysis to provide the data for predictive health analysis. The operator can gather historical information over a specified period and use it in conjunction with the current health data (retrieved from the hot path) to spot trends that might soon cause health issues. In these cases, it might be necessary to raise an alert so that corrective action can be taken.

Diagnose issues

Diagnosis requires the ability to determine the cause of faults or unexpected behavior, including performing root cause analysis. The information that's required typically includes:

- Detailed information from event logs and traces, either for the entire system or for a specified subsystem during a specified time window.
- Complete stack traces resulting from exceptions and faults of any specified level that occur within the system or a specified subsystem during a specified period.
- Crash dumps for any failed processes either anywhere in the system or for a specified subsystem during a specified time window.
- Activity logs recording the operations that are performed either by all users or for selected users during a specified period.

Analyzing data for troubleshooting purposes often requires a deep technical understanding of the system architecture and the various components that compose the solution. As a result, a large degree of manual intervention is often required to interpret the data, establish the cause of problems, and recommend an appropriate strategy to correct them. It might be appropriate simply to store a copy of this information in its original format and make it available for cold analysis by an expert.

Next steps

Visualization

Visualize data and raise alerts

Article • 11/30/2022

An important aspect of any monitoring system is the ability to present the data in such a way that an operator can quickly spot any trends or problems. Also important is the ability to quickly inform an operator if a significant event has occurred that might require attention.

Data presentation can take several forms, including visualization by using dashboards, alerting, and reporting.

Use dashboards to visualize data

The most common way to visualize data is to use dashboards that can display information as a series of charts, graphs, or some other illustration. These items can be parameterized, and an analyst can select the important parameters, such as the time period, for any specific situation.

Dashboards can be organized hierarchically. Top-level dashboards can give an overall view of each aspect of the system but enable an operator to drill down to the details. For example, a dashboard that shows the overall disk I/O for the system should allow an analyst to view the I/O rates for each individual disk to determine whether one or more specific devices account for a disproportionate volume of traffic. Ideally, the dashboard should also display related information, such as the source of each request (the user or activity) that's generating this I/O. This information can then be used to determine whether (and how) to spread the load more evenly across devices, and whether the system would perform better if more devices were added.

A dashboard may also use color-coding or some other visual cues to indicate values that appear anomalous or that are outside an expected range. Using the previous example:

- A disk with an I/O rate that's approaching its maximum capacity over an extended period (a hot disk) can be highlighted in red.
- A disk with an I/O rate that periodically runs at its maximum limit over short periods (a warm disk) can be highlighted in yellow.
- A disk that's showing normal usage can be displayed in green.

For a dashboard system to work effectively, it must have the raw data to work with. If you're building your own dashboard system, or using a dashboard developed by another organization, you must understand the following concepts:

- Which instrumentation data do you need to collect?
- At what levels of granularity?
- How should you format data for the dashboard to consume?

A good dashboard doesn't only display information, it also enables an analyst to pose improvised questions about that information. Some systems provide management tools that an operator can use to complete these tasks and explore the underlying data. Instead, depending on the repository that's used to hold this information, it may be possible to query this data directly, or import it into tools such as Microsoft Excel for further analysis and reporting.

> **Note**
>
> You should restrict access to dashboards to authorized personnel, because this information may be commercially sensitive. You should also protect the underlying data for dashboards to prevent users from changing it.

Reporting

Reporting is used to generate an overall view of the system. It may incorporate historical data and current information. Reporting requirements fall into two broad categories: operational reporting and security reporting.

Operational reporting typically includes the following aspects:

- Aggregating statistics that you can use to understand resource utilization of the overall system or specified subsystems during a specified time window.
- Identifying trends in resource usage for the overall system or specified subsystems during a specified period.
- Monitoring exceptions that have occurred throughout the system or in specified subsystems during a specified period.
- Determining the efficiency of the application for the deployed resources, and understanding whether the volume of resources, and their associated cost, can be reduced without affecting performance unnecessarily.

Security reporting tracks customers' use of the system. It can include:

- *Auditing user operations*: This method requires recording the individual requests that each user completes, together with dates and times. The data should be structured to enable an administrator to quickly reconstruct the sequence of operations that a user completes over a specified period.

- *Tracking resource use by user*: This method requires recording how each request for a user accesses the various resources that compose the system, and for how long. An administrator can use this data to generate a utilization report, by user, over a specified period, possibly for billing purposes.

In many cases, batch processes can generate reports according to a defined schedule. Latency isn't normally an issue. Batch processes should also be available for generation on a spontaneous basis, if needed. As an example, if you are storing data in a relational database, such as Azure SQL Database, you can use a tool such as SQL Server Reporting Services to extract and format data, and present it as a set of reports.

Next steps

More information: Alerting

Alerting for operations

Article • 03/15/2023

Alerting is the process of generating notifications when a significant event is detected after analyzing instrumentation data. It's an important part of any system that makes performance, availability, and privacy guarantees to the users where an action is required immediately.

To ensure that the system remains healthy, responsive, and secure, it's highly recommended that you set alerts so that operators can respond to them in a timely manner. An alert can contain enough contextual information to help you quickly get started on diagnostic activities. Alerting can be used to invoke remediation functions such as autoscaling. Alerts can also enable cost-awareness by watching budgets and limits.

Best practices

- Define a process for alert response that identifies the accountable owners and actions.
- Configure alerts for a well-defined scope (resource types and resource groups) and adjust the verbosity to minimize noise.
- Use an automated alerting solution, such as Splunk or Azure Monitor, instead of having people actively look for issues.
- Take advantage of alerts to operationalize remediation processes. For example, automatically create tickets to track issues and resolutions.
- Track the health of Azure services in regions, communication about outages, planned maintenance activities, and other health advisories.

Alert use cases

Here are some common situations for which you might want to set alerts and associated actions:

- **Security events**—If the event logs indicate that repeated authentication or authorization failures are occurring, the system might be under attack and an operator should be informed.
- **Performance metrics**—If a particular performance metric exceeds a specified threshold, the system must quickly respond.
- **Availability information**—If a fault is detected, it might be necessary to quickly restart one or more subsystems, or failover to a backup resource. Repeated faults

in a subsystem might indicate more serious concerns.
- **Cost and usage information**—If usage or cost exceeds the allocated budget, it might be necessary to raise cost awareness to the department with notification that spending has reached a fixed threshold of the quota.

Application alerts

An alerting system should allow customizing alerts. The appropriate values from the underlying instrumentation data can be provided to the alerting system as parameters. This approach enables an operator to filter data and focus on those thresholds or combinations of interest values.

In some cases, the raw instrumentation data can be provided. In other situations, it might be more appropriate to supply aggregated data. For example, an alert can be triggered if the CPU utilization for a node has exceeded `90%` over the past `10` minutes. The details provided to the alerting system should also include any appropriate summary and context information. This data can help reduce the possibility that false-positive events will trip an alert.

Resource health alerts

Health alerts should be configured for specific resource groups and resource types, and should be adjusted to maximize signal to noise ratios. For example, only send a notification when a resource becomes unhealthy as per the defined requirements of the application health model or due to an Azure platform-initiated event.

Consider transient issues when setting an appropriate threshold for resource unavailability, such as configuring an alert for a virtual machine with a threshold of 1 minute for unavailability before an alert is triggered.

Azure Resource Health provides information about the health of individual resources such as a specific virtual machine instance, and is useful when diagnosing unavailable resources.

You can also view resource health in Azure Monitor. Alerts can be defined by using many different data streams such as metric values, log search queries, and activity log events.

Service health alerts

Get a view into the health of Azure services and regions, communications about outages, planned maintenance activities, and other health advisories.

Consider configuring Azure Service Health alerts to operationalize Service Health events. These alerts can provide useful information in interpreting issues that you might have detected through the resource health alerts. You can use Service Health events to decide the best way to respond operationally.

However, Service Health alerts aren't effective in detecting issues because of the associated latencies. There's a 5 minute Service Level Objectives (SLOs) for automated issues, but many issues require manual interpretation for root cause analysis.

Process considerations

Alerts proactively notify and in cases even respond to operational states that deviate from the norm. When such an event occurs, an alert is triggered to notify the accountable teams.

Having **well-defined owners and response playbooks** per alert is vital to optimizing operational effectiveness. Alerts can be set for non-technical notifications. For example, a budget owner should be made aware of capacity issues so that budgets can be adjusted and discussed.

Instead of having teams actively monitor the systems and dashboard, **send reliable alert notifications** to the owners.

The operators might receive alert information by using many delivery channels such as email, text message, a pager device, or push notifications to a mobile app. Many alerting systems support subscriber groups, and all operators who are members of the same group can receive the same set of alerts.

In Azure Monitor, alerts use action groups to notify the owners.

Subscription notification e-mails

Subscription notification e-mails can contain important service notifications or security alerts. Make sure that subscription notification emails are routed to the relevant technical stakeholders.

IT Service Management (ITSM) integration

ITSM systems can help define workflows. You can use those systems to document issues, notify and assign responsible parties, and track issues. For example, operational alerts from the application could be integrated to automatically create new tickets to track resolution.

Azure Monitor supports integrations with third-party ITSM systems. For example, if you have a custom solution that ingests data through an incoming API, this can be engaged with an Azure Monitor action group each time an alert is raised. Many partner integrations are ready to use out of the box.

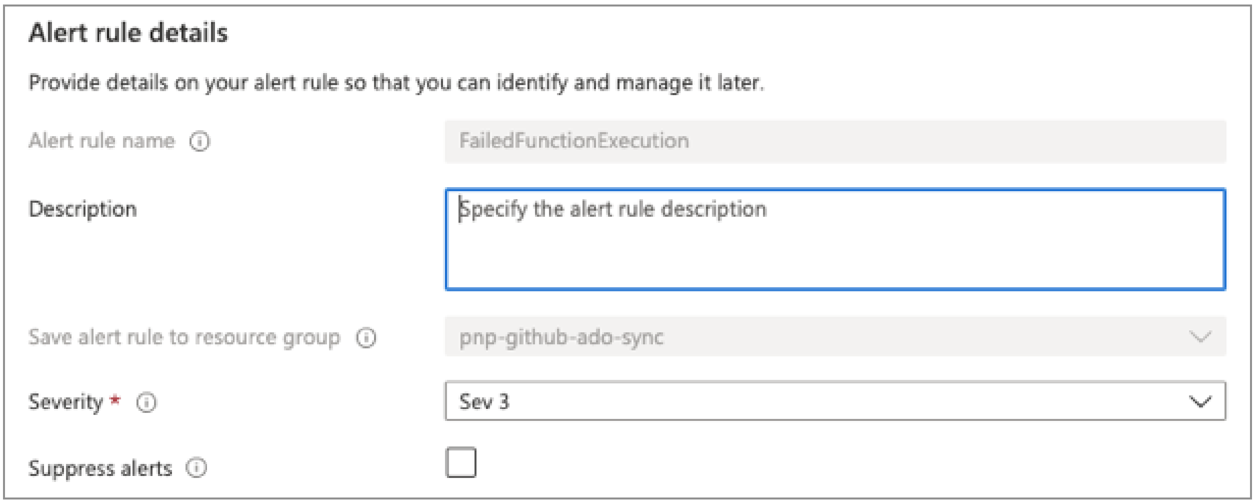
For information on Azure Monitor and ITSM integration, reference IT Service Management Connector Overview.

Alert prioritization

All alerts being treated the same is going to reduce the efficacy of notifications.

When defining alerts, analyze the potential business impact and prioritize accordingly. Prioritizing alerts helps operational teams in cases where multiple events require intervention at the same time. For example, alerts concerning critical system flows might require special attention. When creating an alert, ensure you establish and set the correct priority.

One way of specifying the priority is by using a severity level that indicates how critical a situation is. This image shows this case in Azure Monitor.



Next steps

Return to the operational excellence overview.

Use case: Health monitoring

Related links

- Overview of alerts in Microsoft Azure
- Create and manage action groups

Health monitoring

Article • 11/30/2022

A system is healthy if it's running and can process requests. Health monitoring generates a snapshot of the current health of the system so that you can verify all components are functioning as expected.

Requirements for health monitoring

If any part of the system is unhealthy, you're alerted within a matter of seconds. You'll determine which parts of the system are functioning normally and which parts are experiencing problems. A traffic light system indicates system health:

- Red for unhealthy. The system has stopped.
- Yellow for partially healthy. The system is running with reduced functionality.
- Green for healthy.

A comprehensive health monitoring system enables you to drill down to view the health status of subsystems and components. For example, if the overall system is partially healthy, you can zoom in and determine which functionality is currently unavailable.

Best practices

- Correlate events across all application components.
- Use an Application Performance Management (APM) tool used to collect application level logs.
- Use Application Insights to gather key metrics.
- Collect application logs from different application environments.
- Consider using log levels used to capture different types of application events.
- Capture log messages in a structured format.
- Set out critical application performance targets and non-functional requirements with clarity.
- Identify known gaps in application observability that led to missed incidents or false positives in the past.
- Consider different log aggregation technologies to collect logs and metrics from Azure resources.
- Make logs and metrics available for [critical internal dependencies]#logs-for-internal-dependencies).
- Implement black-box monitoring to measure platform services and the resulting customer experience.

- Implement detailed instrumentation in the application code to better understand the customer experience.
- Apply white-box monitoring to instrument the application with semantic logs and metrics.

Distributed tracing

Trace the execution of user requests to generate raw data to determine which requests have:

- Succeeded
- Failed
- Taken too long

Distributed tracing allows you to build and visualize end-to-end transaction flows for the application. Events coming from different application components or different component tiers of the application should be correlated to build these flows.

For instance, using consistent correlation IDs transferred between components within a transaction achieves end-to-end transaction flows.

Event correlation between application layers allows you to connect tracing data of the complete application stack. You can create a complete picture of where time is spent at each layer through tools that can query the tracing data repositories in correlation to a unique identifier. This unique identifier represents a given transaction that flowed through the system.

Application Performance Management (APM) tools

To successfully maintain an application, it's important to *turn the lights on* to have clear visibility into important metrics, both in real time and historically.

Application Performance Management (APM) tools

An APM technology, such as Application Insights, should be used to manage the performance and availability of the application, aggregating application level logs, and events for later interpretation. With cost in mind, consider the appropriate level of logging required.

Application Insights is an extensible Application Performance Management (APM) service for developers and DevOps professionals to monitor live applications. It automatically detects performance anomalies and includes analytics tools to help users diagnose issues, and to understand what customers do with your application. Application Insights monitors diagnostic trace logs from your application so that you can correlate trace events with requests.

Application Insights allows you to:

- Verify that your application is running correctly.
- Makes application troubleshooting easier.
- Provides custom business telemetry to indicate whether your application is being used as intended.

For more information about how Application Insights helps you monitor applications, reference Monitoring workloads.

Logs and metrics

Logging is essential to understand how an application operates in various environments and what events occur and under which conditions. There are two types of logs: application-generated logs and platform logs.

Application logs

Application logs support the end-to-end application lifecycle. You should collect logs and events across all major application environments. A sufficient degree of separation and filtering should be in place to ensure non-critical environments don't convolute production log interpretation. Corresponding log entries across the application should capture a correlation ID for their respective transactions.

Log levels

Use the following log levels to capture different types of application events across environments:

- **Info**
- **Warning**
- **Error**
- **Debug**

Pre-configure the preceding log levels and apply these levels within relevant environments. Changing log levels includes simple configuration changes to support operational scenarios where it's necessary to raise the log level within an environment.

Log messages

Capture log messages and application events in a structured format, following well-known schema. The structured data type includes machine-readable data points rather than unstructured string types.

Structured data can help you:

- Parse and analyze logs.
- Index and search logs.
- Simplify reporting.

Metrics

Metrics are numerical values that are collected at regular intervals and describe some aspect of a system at a particular time. They reflect the health and usage statistics of your resources.

Analyze health data

Analyzing health data involves quickly indicating whether the system is running through metrics. Hot analysis of the immediate data triggers an alert if a critical component is detected as unhealthy.

For example, the component fails to respond to a consecutive series of pings. You can then take the appropriate corrective action. For more information about analyzing data, reference Analyze monitoring data for cloud applications.

Performance targets and non-functional requirements (NFRs)

Application-level metrics should include end-to-end transaction times of key technical functions, such as:

- Database queries.
- Response times for external API calls.
- Failure rates of processing steps, and so on.

To assess fully the health of key scenarios in the context of targets and NFRs, correlate application log events, such as user login, across critical system flows.

Gaps in application monitoring

Known gaps in application observability lead to missed incidents and false positives.

> **Tip**
>
> *What you can't see, you can't measure. What you can't measure, you can't improve.*

Platform logs

Platform logs provide detailed diagnostic and auditing information for the infrastructure, and resources they depend on. Monitoring your platform includes collecting rich metrics and logs to verify the state of your complete infrastructure, and to react promptly if there are any issues.

Application Insights or a full-stack monitoring service like Azure Monitor can help you keep tabs on your entire landscape.

For more information, reference Monitoring workloads.

Activity logs and diagnostic settings

Activity logs provide audit information about when a resource is modified, such as when a virtual machine is started or stopped. Such information is useful for the interpretation and troubleshooting of operational and performance issues because it provides transparency around configuration changes.

> **Tip**
>
> We recommend collecting and aggregating activity logs.

Log aggregation technologies

Log aggregation technologies, such as Azure Log Analytics or Splunk, should be used to collate logs and metrics across all application components for later evaluation. Resources may include Azure IaaS and PaaS services, and third-party appliances such as firewalls or anti-malware solutions used in the application. For instance, if Azure Event Hub is used, the **Diagnostic Settings** should be configured to push logs and metrics to the data sink. Understanding usage helps with right-sizing of the workload, but extra costs for logging should be accepted and included in the cost model.

All application resources should be configured to route diagnostic logs and metrics to the chosen log aggregation technology. Use Azure Policy to ensure the consistent use of diagnostic settings across the application and to enforce the configuration you want for each Azure service.

Logs for internal dependencies

To build a robust application health model, it's vital to have visibility into the operational state of critical internal dependencies, such as a shared Network Virtual Appliance (NVA) or ExpressRoute connection.

Black-box monitoring

Black-box monitoring tests externally visible application behavior without knowledge of the internals of the system. This type of monitoring is a common approach to measuring customer-centric SLIs, SLOs, and SLAs.

For more information, reference Azure Monitor.

Instrumentation

Instrumentation of your code allows precise detection of underperforming pieces when you apply load or stress tests. It's critical to have this data available to improve and identify performance opportunities in the application code. Use Application Performance Monitoring (APM) tools, such as Application Insights, to manage the performance and availability of the application, along with aggregating application-level logs, and events for later interpretation.

For more resources about instrumentation, reference Monitor performance.

White-box monitoring

Application-level metrics and logs, such as current memory consumption or request latency, should be collected from the application to inform a health model, detect, and predict issues.

For more information, reference Instrumenting an application with Application Insights and Instrument an application.

Next steps

Usage monitoring

Usage monitoring

Article • 11/30/2022

Usage monitoring tracks how the features and components of an application are used.

This article describes how you can use the data gathered from usage monitoring to gain insight into operational events that affect your application and workloads.

Benefits of usage monitoring

The following list explores the use cases for data gathered from usage monitoring:

- Determine which features you use heavily and determine any potential hotspots in the system. High-traffic elements may benefit from functional partitioning or even replication to spread the load more evenly. You can use this information to figure out which features you don't use often and possible candidates for retirement, or replacement in a future version of the system.
- Collect information about the operational events of the system under normal use. For example, in an e-commerce site, you can record the statistical information about the number of transactions and the volume of customers that are responsible for them. You can use this information for capacity planning as the number of customers grows.
- Detect user satisfaction with the performance or functionality of the system. For example, if a large number of customers in an e-commerce system regularly abandon their shopping carts, this behavior may mean there's a problem with the checkout functionality.
- Generate billing information. An application or service may charge customers for the resources they use.
- Enforce quotas. If a user exceeds their paid quota of processing time or resource usage during a specific period, the system can limit their access or throttle processing.

Requirements for usage monitoring

To analyze system usage, you'll need monitoring information that includes:

- The number of requests that each subsystem processes and directs to each resource.
- The work that each user does.
- The volume of data storage that each user occupies.

- The resources that each user accesses.

Requirements for data collection

You can track usage at a relatively high level. Usage tracking can note the start and end times of each request and the nature of the request, such as read, write, and so on, depending on the resource in question. Retrieve this information through the following ways:

- Trace user activity.
- Capture performance counters that measure the usage for each resource.
- Monitor each users' resource consumption.

For accounting purposes, you'll want to identify which users are responsible for doing which operations, and the resources that these operations use. The gathered information should be detailed enough for accurate billing.

Next steps

Issue tracking

Issue tracking

Article • 11/30/2022

If unexpected events occur in the system, customers and other users may report these issues.

Issue tracking involves:

- Managing issues.
- Associating issues with efforts to resolve underlying problems in the system.
- Informing customers of possible resolutions.

Requirements for issue tracking

You can often track issues using a separate system that lets you record and report the details of problems that users report. These details can include:

- Tasks the user was doing.
- Symptoms of the problem.
- Sequence of events.
- Error or warning messages.

Requirements for data collection

The user who initially reported the issue is considered the primary data source. This user can provide more information, such as:

- A crash dump, if the application includes a component that runs on the user's desktop.
- A screenshot.
- The date and time the error occurred.
- The user's location.

You can use this information to help debug and create a backlog for future software releases.

Analyze data

Consider the following scenarios when you analyze issue-tracking data:

- Different users may report the same problem. The issue-tracking system should associate common reports.
- Record the debugging progress against each issue report.
- Inform customers of the solution when you've resolved the issue.
- If a user reports an issue that has a known solution in the issue-tracking system, inform the user of the solution immediately.

Next steps

Tracing and debugging

Tracing and debugging

Article • 11/30/2022

When a user reports an issue, the user is often aware only of the immediate effect that the issue has on their operations. The user can only report the results of their own experience back to the person who is responsible for maintaining the system. These experiences are a visible symptom of one or more fundamental problems.

Root cause analysis

In many cases, an analyst must dig through the chronology of the underlying operations to establish the root cause of the problem. This process is called a *root cause analysis*.

A root cause analysis may uncover inefficiencies in application design. In these situations, you can try to rework the affected elements and deploy them as part of a later release. This process requires careful control, and you should monitor the updated components closely.

Requirements for tracing and debugging

For tracing unexpected events and other problems, consider the following requirements:

- Monitoring data must provide enough information to enable an analyst to trace the origin of an issue and reconstruct the sequence of events that lead up to the issue.
- Data must be sufficient for the analyst to identify a root cause.
- A root cause enables the developer to make the necessary changes to prevent the issue from recurring.

Requirements for data collection

Troubleshooting involves the following data collection requirements:

- Trace all methods and their parameters used in an operation to create a model that shows the logical *flow* through the system when a customer makes a specific request.
- Capture and log exceptions, and warnings that the system generates because of this flow.

To support debugging, the system should provide the following data:

- Hooks that enable you to capture `state` information at critical points in the system.
- Step-by-step information as selected operations continue.

> **Note**
>
> Capturing detailed data can cause extra load on the system and should be a temporary process. Only capture detailed data when an unusual series of events occur, which are difficult to replicate. Alternately, only capture detailed data when you're monitoring a new release to ensure that new elements in the system function as expected.

Next steps

Auditing

Auditing

Article • 11/30/2022

Depending on the application, there may be legal requirements for auditing users' operations and recording all data access. Auditing can provide evidence that links customers to specific requests. Affirming validity is an important factor in many online business systems to help maintain trust between the customer and the business responsible for the application, or service.

Requirements for auditing

An analyst can trace the sequence of business operations that users perform so that you can reconstruct users' actions. Tracing the sequence of operations may be necessary as a matter of record, or as part of a forensic investigation.

Audit information is highly sensitive. This information includes data that identifies the users of the system and the tasks that they're doing. Reports contain sensitive audit information available only to trusted analysts. An analyst can generate a range of reports. For example, reports may list the following activities:

- All users' activities occurring during a specified time frame.
- The chronology of a single user's activity.
- The sequence of operations performed against one or more resources.

Requirements for data collection

The primary sources of auditing information can include:

- The security system that manages user authentication.
- Trace logs that record user activity.
- Security logs that track all network requests.

Regulatory requirements may dictate the format of the audit data and the way it's stored. For example, it may not be possible to clean the data in any way. It must be recorded in its original format. Access to the data repository must be protected to prevent tampering.

Analyze audit data

An analyst must access all the raw data in its original form. Aside from the common audit report requirement, the tools for analyzing this data are specialized and external to the system.

Next steps

DevOps Checklist

Operational Excellence patterns

Article • 05/18/2023

Cloud applications run in a remote datacenter where you don't have full control of the infrastructure or, in some cases, the operating system. This situation can make management and monitoring more difficult than an on-premises deployment. Applications must expose runtime information that administrators and operators can use to manage and monitor the system. Applications must also support changing business requirements and customization without requiring the application to be stopped or redeployed.

Pattern	Summary
Ambassador	Create helper services that send network requests on behalf of a consumer service or application.
Anti-corruption Layer	Implement a façade or adapter layer between a modern application and a legacy system.
External Configuration Store	Move configuration information out of the application deployment package to a centralized location.
Gateway Aggregation	Use a gateway to aggregate multiple individual requests into a single request.
Gateway Offloading	Offload shared or specialized service functionality to a gateway proxy.
Gateway Routing	Route requests to multiple services using a single endpoint.
Health Endpoint Monitoring	Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals.
Sidecar	Deploy components of an application into a separate process or container to provide isolation and encapsulation.
Strangler Fig	Incrementally migrate a legacy system by gradually replacing specific pieces of functionality with new applications and services.

Performance efficiency documentation

Apply the principles of scalability and testing to your architecture to meet users' demands efficiently.

Key points

QUICKSTART

Overview

Principles

Performance efficiency checklist

Test checklist

Optimize checklist

Performance efficiency patterns

Tradeoffs

TRAINING

Performance efficiency

VIDEO

Azure SQL Capacity Planning: Overview

Design for performance efficiency

GET STARTED

Design applications for performance

Design applications for efficiency

Design applications for scalability

Design applications for capacity

CONCEPT

Consider team expertise

Choose the right data storage

Reduce response time with asynchronous programming

Plan for growth

Scale out rather than scaling up

TRAINING

Advance resilience through chaos engineering and fault injection

VIDEO

Autoscale applications on Kubernetes with Kubernetes Event-Driven Autoscaling (KEDA)

Test for performance efficiency

OVERVIEW

Performance testing

Load testing

Stress testing

Multiregion testing

Testing tools

CONCEPT

Establish baselines

Configure the environment based on testing results

Consider caching data

Use a content delivery network

TRAINING

Identify performance bottlenecks with Azure Load Testing

Identify performance regressions through automated load testing

VIDEO

Scale your cloud app with Azure Cache for Redis

Scale your cloud app with Azure Cache for Redis

Monitor for performance efficiency

OVERVIEW

Monitor distributed architectures

Consider application profiling

Analyze infrastructure metrics and logs

Integrate performance data

Monitor performance for scalability and reliability

CONCEPT

Collect application logs

View platform metrics

Correlate and evaluate data

Consider cost implications

VIDEO

Azure Application Insights Profiler

Optimize for performance efficiency

OVERVIEW

Cache data for performance optimization

Partition data for performance optimization

Sustain performance efficiency over time

CONCEPT

Consider capacity

Determine acceptable performance optimization

Optimize autoscaling

Partition critical workloads

VIDEO

Optimize costs of your backups with Azure Backup

Performance tools and services

REFERENCE

Application Insights

Azure Autoscale

Azure Blob Storage

Azure Content Delivery Network (CDN)

Azure Cosmos DB

Azure Front Door

Azure Load Testing

Azure Monitor

B-series virtual machines

Microsoft Well-Architected Review

Performance APIs

REFERENCE

Azure CDN REST API

Azure Monitor REST API

Azure Cache for Redis API

Azure Storage REST API

Overview of the performance efficiency pillar

Article • 05/30/2023

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. Before cloud computing became popular, when it came to planning how a system would handle increases in load, many organizations intentionally provisioned oversized workloads to meet business requirements. This decision made sense for on-premises environments because it ensured *capacity* during peak usage. Capacity reflects resource availability (CPU and memory). Capacity was a major consideration for processes that would be in place for many years.

Just as you need to anticipate increases in load for on-premises environments, you need to expect increases in cloud environments to meet business requirements. One difference is that you might no longer need to make long-term predictions for expected changes to ensure you'll have enough capacity in the future. Another difference is in the approach used to manage performance.

To assess your workload using the tenets found in the Microsoft Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.

To boost performance efficiency, you should watch Performance Efficiency: Fast & Furious: Optimizing for Quick and Reliable VM Deployments.

Articles

The performance efficiency pillar covers the following articles to help you effectively scale your workload:

Performance efficiency article	Description
Performance efficiency checklist	Review your application architecture to ensure your workload scales to meet the demands placed on it by users in an efficient manner.
Performance principles	Learn about the principles that can guide you in your overall strategy for improving performance efficiency.

Performance efficiency article	Description
Design for performance	Review your application architecture from a performance design standpoint.
Consider scalability	Plan for growth by understanding your current workloads.
Plan for capacity	Plan to scale your application tier by adding extra infrastructure to meet demand.
Performance monitoring checklist	Monitor services and check the health state of current workloads to maintain overall workload performance.
Performance patterns	Implement design patterns to build more performant workloads.
Tradeoffs	Consider tradeoffs between performance optimization and other aspects of the design, such as reliability, security, cost efficiency, and operability.

Next steps

Reference the performance efficiency principles intended to guide you in your overall strategy.

Principles

Performance efficiency principles

Article • 11/22/2022

Performance efficiency is the ability of your workload to adjust to changes in demands placed on it by users in an efficient manner. These principles are intended to guide you in your overall strategy for improving performance efficiency.

Design for horizontal scaling

Horizontal scaling allows for elasticity. Instances are added (scale-out) or removed (scale-in) in response to changes in load. Scaling out can improve resiliency by building redundancy. Scaling in can help reduce costs by shutting down excess capacity.

Approach	Benefit
Define a capacity model according to the business requirements	Test the limits for **predicted and random spikes and fluctuations** in load to make sure the application can scale. Factor in the SKU service limits and regional limits so that application scales as expected if there's a regional failure.
Use PaaS offerings	Take advantage of the built-in capabilities that **automatically trigger scaling** operations instead of investing in manual scaling efforts that often require custom implementations and can be error prone.
Choose the right resources and right-size	Determine if the resources can support the anticipated load. Also, justify the cost implications of the choices.
Apply strategies in your design early	Accelerate adoption without significant changes. For example, **strive for stateless application** and store state externally in a database or distributed cache. **Use caching** where possible, to minimize the processing load.

An alternate approach is vertical scaling (scale up). However, you eventually may reach a limit where there isn't a larger system, and you can't scale up anymore. At that point, any further scaling must be horizontal. So it's good practice to employ a scale-out architecture early on.

Shift-left on performance testing

Test early and test often to catch issues early.

Approach	Benefit

Approach	Benefit
Run load and stress tests	Measure the application's performance under **predetermined amounts of load** and also the **maximum load** your application and its infrastructure can withstand.
Establish performance baselines	Determine the **current efficiency** of the application and its supporting infrastructure. You'll be able to identify bottlenecks early before it worsens with load. Also, this strategy can lead to **strategies for improvements** and determine if the application is meeting the business goals.
Run the test in the continuous integration (CI) build pipeline.	Detect issues early. Any development effort must go through continuous performance testing to make sure changes to the codebase doesn't negatively affect performance.

Continuously monitor for performance in production

Observe the system holistically to evaluate the overall health of the solution. Capture the test results not only in dev/test environment but also in production. Monitoring and logging in production can help identify bottlenecks and opportunities for improvement.

Approach	Benefit
Monitor the health of the entire solution	Know about the scalability *and* resiliency of the infrastructure, application, and dependent services. Gather and review key performance counters regularly.
Capture data from repeatable processes	Evaluate the metrics over time that would allow for autoscaling with demand. For reliability, look for early warning signs that might require proactive intervention.
Reevaluate the needs of the workload continuously	Identify improvement opportunities with resolution planning. This effort may require updated configurations and deprecations in favor of more-appropriate solutions.

Next section

Use this checklist to review your application architecture from a performance design standpoint.

Design checklist

Related links

- Performance efficiency impacts the entire architecture spectrum and is interrelated with other pillars of the Microsoft Azure Well-Architected Framework.
- Assess your workload using the Microsoft Azure Well-Architected Review tool.

Challenges of monitoring distributed architectures

Article • 05/30/2023

Most cloud deployments are based on distributed architectures where components are distributed across various services. Troubleshooting monolithic applications often requires only one or two lenses—the application and the database. With distributed architectures, troubleshooting is complex and challenging because of various factors. This article describes some of those challenges.

Key points

- ✓ The team might not have the expertise across all the services used in an architecture.
- ✓ Uncovering and resolving bottlenecks by monitoring all of your services and their infrastructure is complex.
- ✓ Antipatterns in design and code causes issues if the application is under pressure.
- ✓ Resilience in any service might impact your application's ability to meet current load.

Team expertise

Distributed architectures require many areas of expertise. To adequately monitor performance, it's critical that telemetry is captured throughout the application, across all services, and is rich. Also, your team should have the necessary skills to troubleshoot all services used in the architecture. When making technology choices, it's advantageous and simple to choose a service over another because of the team's expertise. As the collective skill set grows, you can incorporate other technologies.

Scaling issues

For monolithic applications, scale is two-dimensional. An application usually consists of a group of application servers, some web front ends (WFEs), and database servers. *Uncovering* bottlenecks is simpler but *resolving* them can require considerable effort. For distributed applications, complexity increases exponentially in both aspects for performance issues. You must consider each application, its supporting service, and the latency between all the application layers.

Performance efficiency is a complex mixture of applications and infrastructure (IaaS and PaaS). First, you must ensure that all services can scale to support the expected load and that one service doesn't cause a bottleneck. Second, while performance testing, you might realize that certain services scale under different load conditions as opposed to scaling all services uniformly. Monitoring all of your services and their infrastructure can help fine-tune your application for optimal performance.

For more information about monitoring for scalability, see Monitor performance for scalability and reliability.

Antipatterns in design

Antipatterns in design and code are a common cause for performance problems when an application is under pressure. For example, an application behaves as expected during performance testing. However, when it's released to production and starts to handle live workloads, performance decreases. Problems such as rejecting user requests, stalling, or throwing exceptions might arise. To learn how to identify and fix these antipatterns, see Performance antipatterns for cloud applications.

Fault tracking

If a service in the architecture fails, how does it affect your application's overall performance? Is the error transient, allowing your application to continue to function. Or, will the application experience a critical failure? If the error is transient, does your application experience a decrease in performance?

Resiliency plays a significant role in performance efficiency because the failure of any service can impact your application's ability to meet your business goals and scale to meet current load. Chaos testing—the introduction of random failures within your infrastructure—against your application can help determine how well your application continues to perform under varying stages of load.

For more information about reliability impacts on performance, see Monitor performance for scalability and reliability.

Next

Design scalable Azure applications

Community links

To learn more about chaos testing, see Advancing resilience through chaos engineering and fault injection .

Related links

- Performance antipatterns for cloud applications
- Monitor performance for scalability and reliability

> Back to the main article

Design applications for performance

Article • 05/18/2023

Application design is critical to ensuring performance efficiency as load increases. This article gives you insights on the most important aspects of designing applications for performance. For more performance efficiency design related articles, see:

- Design applications for efficiency
- Design applications for scale
- Design applications for capacity

Choose the right storage

The overall design of the storage tier can greatly affect an application's performance and scalability. The Azure Storage platform is designed to be massively scalable to meet the data storage and performance needs of modern applications.

Data services in the Azure Storage platform include:

- Azure Blob, a massively scalable object store for text and binary data, which includes support for big data analytics through Azure Data Lake Analytics.
- Azure Files, managed file shares for cloud or on-premises deployments.
- Azure Queue, a messaging store for reliable messaging between application components.
- Azure Tables, a NoSQL store for schemaless storage of structured data.
- Azure Disks, block-level storage volumes for Azure virtual machines (VMs).

Each service is accessed through an Azure Storage account. To get started, see Create a storage account.

Most cloud workloads adopt the *polyglot* persistence approach. Instead of one data store service, a mix of technologies is used. Your application might require more than one type of data store. For guidance about when to use different storage types, see Sample scenarios for Azure Storage services.

Choose the right databases

The choice of database can affect an application's performance and scalability. Database reads and writes involve network calls and storage input and output (I/O), both of which are expensive. Choosing the right database service to store and retrieve data is therefore critical to ensure application performance. Azure has many database services to fit most needs. There are also third-party options from Azure Marketplace that can be considered.

To help choose a database type, determine whether the application's storage requirements fit a relational design (SQL), or a key-value, document, or graph design (NoSQL). Some applications might use both a SQL and a NoSQL database for different storage needs.

For a detailed description of NoSQL and relational databases, see Understand distributed NoSQL databases. See Select an Azure data store for your application to help find appropriate managed data storage solutions.

Relational databases

Use a relational database when strong consistency guarantees are important. All changes are atomic, and transactions always leave the data in a consistent state. A relational database can't usually scale out horizontally without sharding the data, and implementing manual sharding can be time consuming. Also, the data in relational database must be normalized, which isn't appropriate for every data set.

If a relational database is optimal, Azure offers several platform as a service (PaaS) options that fully manage database hosting and operations. Azure SQL can host single databases or multiple databases through Azure SQL Managed Instance. Azure database offerings span performance, scale, size, reliability, disaster recovery, and migration compatibility requirements. Azure offers the following PaaS relational database services:

- Azure SQL Database
- Azure Database for MySQL
- Azure Database for PostgreSQL
- Azure Database for MariaDB

NoSQL databases

Use a NoSQL database when application performance and availability are more important than strong consistency. NoSQL databases are ideal for handling large, unrelated, indeterminate, or rapidly changing data. NoSQL databases also have tradeoffs. For more information, see Understand distributed NoSQL database challenges.

Azure provides two managed services that optimize for NoSQL solutions, Azure Cosmos DB and Azure Cache for Redis. For document and graph databases, Azure Cosmos DB offers extreme scale and performance.

Choose the right VM sizes

Choose VM sizing requirements based on your needs. Choosing the wrong VM size can result in capacity issues as VMs approach their limits, or can lead to unnecessary costs.

To choose the right VM size, consider your workloads, number of CPUs, RAM capacity, disk size, and speed according to business requirements. For more information about Azure VM sizes and their purposes, see Sizes for virtual machines in Azure.

Azure offers the following categories of VM sizes, each designed to run different workloads.

- General-purpose provide balanced CPU-to-memory ratio ideal for testing and development, small to medium databases, and low to medium traffic web servers.
- Memory optimized offer a high memory-to-CPU ratio good for relational database servers, medium to large caches, and in-memory analytics.
- Compute optimized have a high CPU-to-memory ratio good for medium traffic web servers, network appliances, batch processes, and application servers.
- GPU optimized are available with single, multiple, or fractional GPUs designed for compute-intensive, graphics-intensive, and visualization workloads.
- High performance compute are designed to deliver leadership-class performance, scalability, and cost efficiency for various real-world high performance computing (HPC) workloads.
- Storage optimized offer high disk throughput and I/O ideal for big data, SQL and NoSQL databases, data warehousing, and large transactional databases. Example databases include Cassandra, MongoDB, Cloudera, and Redis.

Use microservices architectures

Microservices are a popular architectural style for building applications that are resilient, highly scalable, independently deployable, and able to evolve quickly. A microservices architecture consists of a collection of small, autonomous services. Each service is self-contained and implements a single business capability.

Breaking up larger entities into small discrete pieces doesn't ensure sizing and scaling capabilities. Application logic must be written to control these capabilities. One benefit of microservices is that they can be scaled independently. Independent scaling lets you scale out subsystems that require more resources without scaling the entire application.

Another microservices benefit is fault isolation. If an individual microservice becomes unavailable, it doesn't disrupt the entire application. Upstream microservices should be designed to handle faults correctly, for example by implementing circuit breaking. To learn more about the benefits of microservices, see Microservices architecture design benefits.

Building with microservices has challenges for development and testing. Writing a small service that relies on other dependent services requires a different approach than writing a traditional monolithic or layered application. Existing tools aren't always designed to work with service dependencies. Refactoring across service boundaries can be difficult. It's also challenging to test service dependencies, especially when the application is evolving quickly. For more information about the potential drawbacks of a microservices architecture, see Microservices architecture design challenges.

Dynamic service discovery for microservices applications

In a microservices application, many separate services or instances of services in play must receive instructions on who to contact and other configuration information. Hard coding this information is flawed. Through service discovery, a service instance can spin up and dynamically discover the configuration information it needs to become functional without having the information hard coded.

Use an orchestration platform designed to execute and manage individual services, such as Kubernetes or Service Fabric. Individual services can be right sized, scaled up, scaled down, and dynamically configured to match user demand. An orchestrator like Azure Kubernetes Service (AKS) or Azure Service Fabric can pack a higher density of services onto a single host, which allows for more efficient resource utilization.

Both platforms provide built-in services for executing, scaling, and operating a microservices architecture. These services include discovery and finding where a particular service is running. AKS supports pod autoscaling and cluster autoscaling. For more information, see Advanced AKS microservices architecture autoscaling. Service Fabric architecture takes a different approach to scaling for stateless and stateful services. For more information, see Azure Service Fabric scaling services.

> **Tip**
>
> Decomposing an application into microservices is a level of decoupling that is an architectural best practice when appropriate. A microservices architecture can also bring some challenges. The design patterns in **Design patterns for microservices** can help mitigate these challenges.

Use connection pooling

Establishing a connection to a database is an expensive operation that requires creating an authenticated network connection to the remote database server. Database connections are especially expensive for applications that open new connections frequently. Connection pooling reuses existing connections and avoids the expense of opening a new connection for each request. Connection pooling reduces connection latency and enables higher database throughput (transactions per second) on the server.

Pool size limits

Azure limits the number of network connections for a VM or Azure App Service instance. Exceeding this limit causes connections to be slowed down or terminated. With connection pooling, a fixed set of connections is established at startup time and maintained. In many cases, a default pool size might consist only of a small handful of connections that perform quickly in basic test scenarios.

The default pool size might become a bottleneck under scale when the pool is exhausted. It's a best practice to establish a pool size that maps to the number of concurrent transactions supported on each application instance.

Each database and application platform has slightly different requirements for the right way to set up and use the pool. For a .NET code example that uses SQL Server and Azure SQL Database, see SQL Server connection pooling. In all cases, testing is important to ensure a connection pool is properly established and works as designed under load.

> **Tip**
>
> Choose a pool size that uses the same number of concurrent connections. Choose a size that can handle more than the existing connections so you can quickly handle a new incoming request.

Integrated security and connection pools

Integrated security is a single unified solution that uses a set of common policies and configuration settings to protect every service that a business runs. Integrated security reduces scaling, provisioning, and management issues, including higher costs and complexity, and increases control and overall security.

However, sometimes you might not want to use connection pooling for security reasons. For example, although connection pooling improves the performance of subsequent database requests for a single user, that user can't take advantage of connections made by other users. Connection pooling also results in at least one connection per user to the database server.

Measure your business' security requirements against the advantages and disadvantages of connection pooling. For more information, see Pool fragmentation.

Next steps

Design for efficiency

Design applications for efficiency

Article • 05/18/2023

Making choices that improve performance efficiency is critical to application design. This article describes the most important aspects of designing efficient applications. For more performance efficiency design related guidance, see Design for performance.

Reduce response time with asynchronous programming

The time for a caller to receive a response can range from milliseconds to minutes. During that time, the thread is held by the process until the response comes back or an exception occurs. Holding the thread is inefficient, because no other requests can be processed during the response wait time. An example of multiple requests in flight being inefficient is a bank account where only one resource can operate on the request at a time. Another example is when connection pools can't be shared, so all the requests need separate connections to complete.

Asynchronous processing is an alternative approach that enables a remote service to be executed without waiting and blocking resources on the client. Asynchronous programming is a critical pattern for enabling performance efficiency, and is available in most modern programming languages and platforms.

There are many ways to inject asynchronous programming into an application design. For APIs and services that work across the internet, consider using the asynchronous request-reply pattern. In code, remote calls can be asynchronously executed by using built-in language constructs like `async`/`await` in .NET C#. For more information, see Asynchronous programming with async and await.

The .NET platform has other built-in support for asynchronous programming. For more information, see Task-based asynchronous pattern and Event-based asynchronous pattern.

Process faster by queuing and batching requests

Similar to asynchronous programming, queuing is a scalable mechanism to hand off processing work to a service. Highly scalable queuing services are natively supported in Azure. The queue is a storage buffer located between the caller and the processing service. The queue takes requests, stores them in a buffer, and queues the requests to provide services for the reliable delivery and management of the queued data.

A queue is often the best way to hand off work to a processing service. The processing service receives work by listening on the queue and dequeuing messages. If items to be processed enter too quickly, the queuing service keeps them in the queue until the processing service has available resources and asks for a new message.

The processing service can use dynamic Azure Functions to easily autoscale on demand as the queue builds up to meet the intake pressure. Developing processor logic with Azure Functions to run task logic from a queue is a common, scalable, and cost effective way to use queuing between a client and a processor.

Azure provides some native first-party queueing services. These services include Azure Queue Storage, a simple queuing service based on Azure Storage, and Azure Service Bus, a message broker service that supports transactions and reduced latency. Many third-party options are also available through Azure Marketplace.

To learn more about queue-based load leveling, see Queue-based load leveling pattern. Also see Storage queues and Service Bus queues compared and contrasted.

Optimize with data compression

A well-known optimization best practice is to use a compression strategy to compress and bundle web pages or API responses. Compression shrinks the data returned from a page or API back to the browser or client app. Compressing the data returned to clients optimizes network traffic and accelerates the application.

Azure Front Door can do file compression, and .NET has built-in framework support for this technique with *gzip* compression. For more information, see Response compression in ASP.NET Core.

Improve performance with session affinity

If your application is stateful, and data or state is stored locally in the instance of the application, enabling session affinity might improve application performance. When session affinity is enabled, subsequent requests to the application are directed to the same server that processed the first request. If session affinity isn't enabled, subsequent requests are directed to the next available server, depending on the load balancing rules.

Session affinity allows the instance to have some persistent or cached data or context, which can speed subsequent requests. However, if your application doesn't store large amounts of state or cached data in memory, session affinity might decrease your throughput. One host could get overloaded with requests, while others are dormant.

Tip

For information about best practices for stateless services when an application is migrated from old Azure Cloud Services to Azure Service Fabric, see **Migrate an Azure Cloud Services application to Azure Service Fabric**.

Run background jobs to meet integration needs

Many types of applications require background tasks that run independently of the user interface. Examples include batch jobs, intensive processing tasks, and long-running processes such as workflows. Background jobs can be run without requiring user interaction. The application can start the job and continue to process interactive requests from users. For more information, see Background jobs.

Background tasks must perform adequately to ensure that they don't block the application or cause inconsistencies due to delayed operation when the system is under load. Performance can be improved by scaling the compute instances that host the background tasks. For a list of considerations, see Background jobs scaling and performance considerations.

Logic Apps is a serverless pay-per-use consumption service that enables a set of ready-to-use out-of-the-box connectors and a long-running workflow engine to quickly meet cloud-native integration needs. Logic Apps is flexible enough to support scenarios like running tasks or jobs, advanced scheduling, and triggering.

Logic Apps also has advanced hosting options to allow it to run within enterprise restricted cloud environments. Logic Apps can be combined with and complement other Azure services, or can be used independently.

Like all serverless services, Logic Apps doesn't require VM instances to be purchased, enabled, or scaled up and down. Logic Apps scales automatically on serverless platform as a service (PaaS)-provided instances, and you pay only for usage.

Next steps

Design for scaling

Design applications for scaling

Article • 05/18/2023

Scalability, an important aspect of performance efficiency, is the ability of a system to resize to handle changing load. This article describes the types of scaling, how to plan for growth, and how to use platform autoscaling features to manage load.

To achieve scalability, consider how your application scales, and implement platform as a service (PaaS) offerings that have built-in scaling operations. Azure services that *autoscale* can scale automatically to match demand to workload. For example, these services automatically scale out to ensure capacity during workload peaks, and return to normal automatically when the load drops. For more information about autoscaling, see Use autoscaling to manage load.

Understand horizontal and vertical scaling

Two ways an application can scale include *vertical scaling* and *horizontal scaling*. Vertical scaling, or scaling up, increases the capacity of a resource, for example by using a larger virtual machine (VM) size. Horizontal scaling, or scaling out, adds new instances of a resource, such as VMs or database replicas.

An advantage of vertical scaling is that you can scale up without changing the application. But at some point, you hit a limit where you can't scale up anymore. Any further scaling must then be horizontal.

Horizontal scaling has the following significant advantages over vertical scaling:

- **True cloud scale applications can be designed** to run on hundreds or even thousands of nodes, reaching scales that aren't possible on a single node.
- **Horizontal scale is elastic**, so you can add more instances if load increases, or remove instances during quieter periods.
- **Scaling out can be triggered automatically**, either on a schedule or in response to changes in load.
- **Scaling out might be cheaper than scaling up**, because running several small VMs can cost less than running a single large VM.
- **Horizontal scaling can improve reliability** by adding redundancy. If an instance goes down, the application keeps running.

Horizontal scale must be designed into the system. For example, you can scale out VMs by placing them behind a load balancer. But each VM in the pool must handle any client request, so the application must be stateless or store state externally, as in a distributed cache. Managed PaaS services often have horizontal scaling and autoscaling built in. The ease of scaling is a major advantage of using PaaS services.

Simply adding more instances doesn't mean an application can scale. Adding more instances could push a bottleneck somewhere else. For example, scaling a web front end to handle more client requests might trigger lock contentions in the database. You need to consider measures such as optimistic concurrency or data partitioning to enable more throughput to the database.

Always conduct performance and load testing to find potential bottlenecks. Resolving one bottleneck might reveal other bottlenecks elsewhere. The stateful parts of a system, such as databases, are the most common cause of bottlenecks, and require careful design to scale horizontally.

Plan for growth

Use the Performance efficiency checklist to review your design from a scalability standpoint. For more information about how to determine the upper and maximum limits of an application's capacity, see Performance testing.

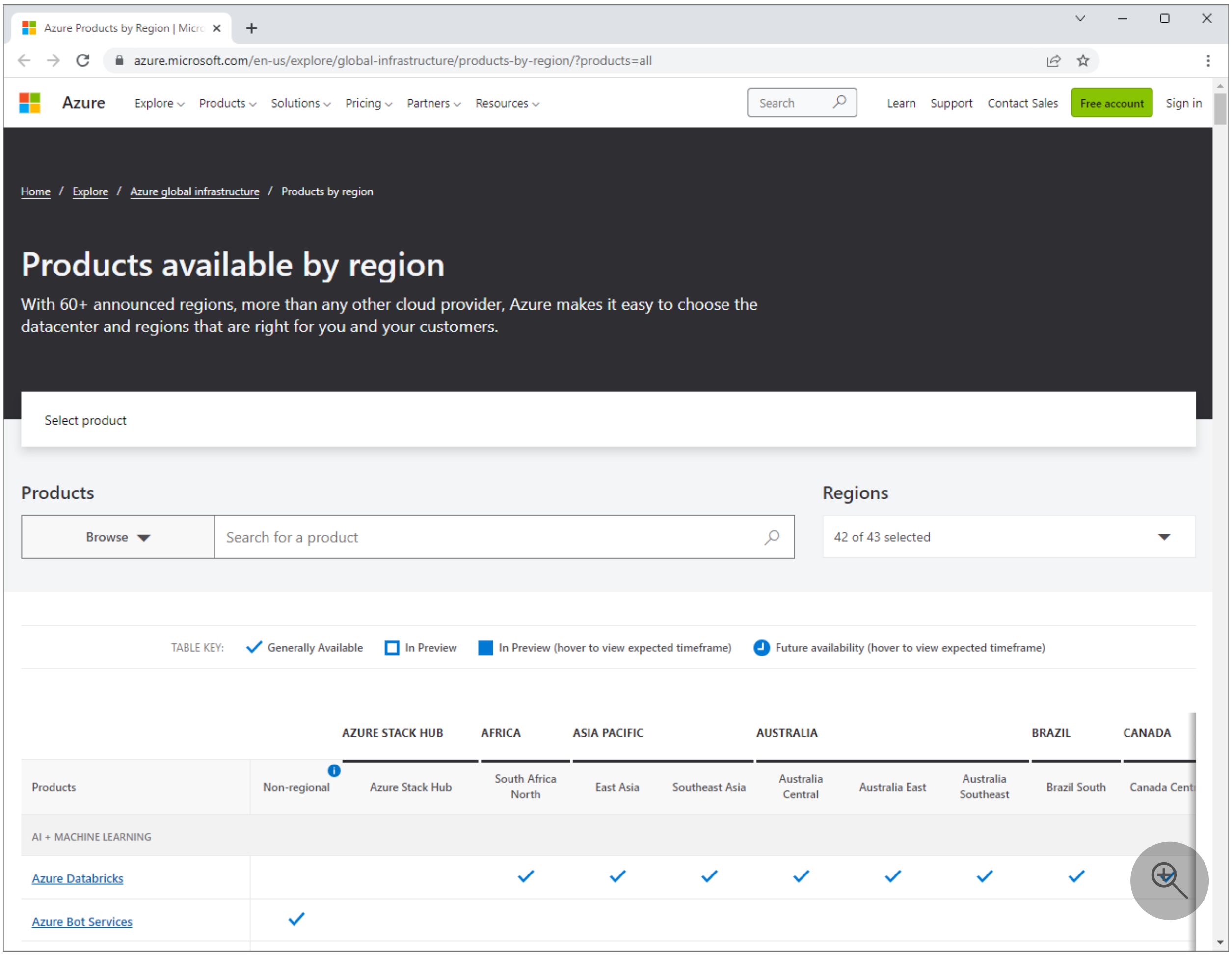
In the cloud, the ability to take advantage of scalability depends on your infrastructure and services. Platforms like Kubernetes were built with scaling in mind. VMs might not scale as easily, although scale operations are possible. With VMs, you might want to plan ahead to avoid scaling infrastructure in the future. Another option is to select a different platform such as virtual machine scale sets.

Planning for growth starts with understanding your current workloads, which can help you anticipate scaling needs based on predictive usage scenarios. An example of a predictive usage scenario is an e-commerce site that recognizes that its infrastructure should scale appropriately for an expected high volume of holiday traffic.

Carry out load tests and stress tests to determine the necessary infrastructure to support predicted spikes in workloads. A good plan includes incorporating a buffer to accommodate random spikes. With scalable platforms, you can predict the current average and peak times for your workload and manage this prediction with payment plans. You pay either per minute or per hour, depending on the service.

Another critical component of planning for scale is to make sure the region that host your application supports the necessary capacity to accommodate the load increase. If you use a multiregion architecture, make sure the secondary regions can also support the increase.

A region might offer a product, but might not have the necessary SKUs to support the predicted load increase, so you need to verify capacity. To verify your regions and available SKUs, first select the product and regions in Products available by region. Then, use the Azure portal to check the SKUs that are available.



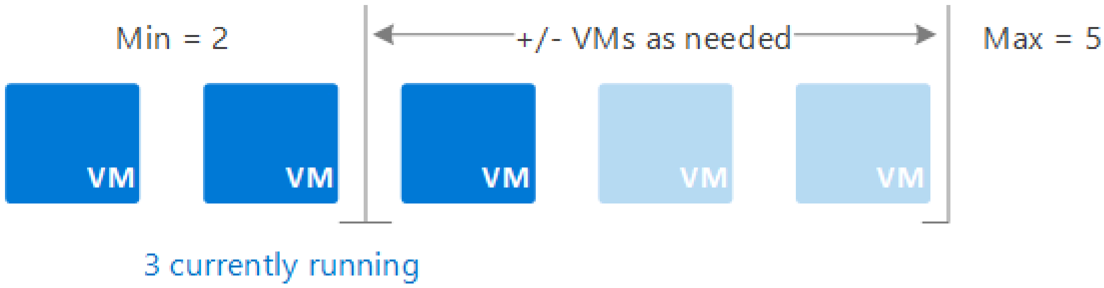
Determine scale units

For each resource, know the upper scaling limits, and use sharding or decomposition if you need to go beyond those limits. Design the application so that it's easily scaled by adding one or more scale units, such as by using the deployment stamps pattern.

Use built-in scaling features or tools to understand which resources need to scale concurrently with other resources. Then use well-defined sets of resources to determine the scale units for the system. For example, adding *X* number of front-end VMs might require *Y* number of extra queues and *Z* number of storage accounts to handle the extra workload. So a scale unit could consist of *X* VM instances, *Y* queues, and *Z* storage accounts.

Use autoscaling to manage load

Autoscaling lets you automatically run the right amount of resources to handle your app load. Autoscaling adds resources, or scales out, to handle increases in load such as seasonal workloads. Autoscaling also removes idle resources, or scales in, to save money during decreases in load, such as nights and weekends for some corporate apps. Autoscaling automatically scales between the minimum and maximum number of instances to run, and automatically adds or removes VMs based on a set of rules.



Understand autoscaling delays

Horizontal scaling changes the number of identical instances, and vertical scaling switches to more or less powerful instances. Scaling operations can be fast, but they take time to complete. It's important to understand how the delay affects the application under load, and whether degraded performance is acceptable.

For more information, see Best practices for autoscale.

Use platform autoscaling features

To get the greatest benefit from autoscaling, follow these practices:

- Use built-in autoscaling features when possible, rather than custom or third-party mechanisms.
- Use scheduled scaling rules where possible, to ensure that resources are available.
- Add reactive autoscaling to the rules where appropriate, to cope with unexpected changes in demand.

> ⓘ **Note**
>
> If your application is explicitly designed to handle the termination of some of its instances, use autoscaling to reduce operational costs. Scale down and scale in resources that are no longer necessary for the load.

For more information, see Autoscaling.

Autoscale compute or memory-intensive applications

CPU or memory-intensive applications can require scaling up to a larger machine SKU with more CPU or memory. Once the demand for CPU or memory decreases, instances can revert back to the original size.

For example, you might have an application that processes images, videos, or music. While scaling out allows the system to process more files simultaneously, it doesn't improve the processing speed for each individual file. Given the process and requirements, it might make sense to scale up a server by adding CPU or memory to quickly process a large media file.

Use Azure services that autoscale

Autoscaling works by collecting resource metrics like CPU and memory usage, and application metrics like requests queued and requests per second. Rules can then be created to add and remove instances depending on how the rule evaluates.

An Azure App Service plan allows autoscale rules to be set for scale out/scale in and scale up/scale down. The App Service autoscaling sample shows how to create an Azure App Service plan, which includes an Azure App Service. Scaling also applies to Azure Automation.

Azure Kubernetes Service (AKS) offers two levels of autoscale:

- **Horizontal autoscale** can be enabled on service containers to add more or fewer pod instances within the cluster.
- **Cluster autoscale** can be enabled on the agent VM instances running an agent node pool to add or remove VM instances dynamically.

The following Azure services also offer autoscaling capability:

- Azure Service Fabric with virtual machine scale sets offers autoscale capabilities for true infrastructure as a service (IaaS) scenarios.
- Azure Application Gateway and Azure API Management are PaaS offerings for ingress services that enable autoscale.
- Azure Functions, Azure Logic Apps, and Azure App Services provide serverless pay-per-use consumption modeling that inherently provides autoscaling capabilities.
- Azure SQL Database is a PaaS platform that changes performance characteristics of a database on the fly and assigns more resources when needed or releases them when not needed. SQL Database allows scaling up/down, read scale-out, and global scale-out/sharding capabilities.

Each service documents its autoscale capabilities. For a general discussion about Azure platform autoscaling, see Overview of autoscale in Azure.

> ⓘ **Note**
>
> If your application doesn't have built-in autoscale ability, or isn't configured to scale out automatically as load increases, its services might fail if they become saturated with user requests. For possible solutions, see **Set up a scaling tool by using Azure Automation and Azure Logic Apps for Azure Virtual Desktop**.

Next steps

Design for capacity

Design applications for capacity

Article • 05/18/2023

This article describes ways you can design applications for capacity to optimize performance efficiency. Azure offers many options to meet capacity requirements while minimizing costs as your business grows.

Scale out rather than scaling up

With cloud technologies, it's often easier, cheaper, and more effective to *scale out*, or add resources, than to *scale up*, or increase the capacity of a resource. If you plan to scale up by increasing the capacity allocated to your hosts, you reach a limit where it becomes cost-prohibitive to scale any further. Scaling up also often requires downtime for servers to reboot. Instead, plan to scale your application tier by adding extra infrastructure to meet demand. Be sure to remove the resources when they're no longer needed.

Prepare infrastructure for large-scale events

Large-scale application design takes careful planning and possibly complex implementation. Work with your business and marketing teams to prepare for large-scale events like sports matches, sales, or marketing campaigns. You can anticipate sudden spikes in traffic caused by these events to prepare infrastructure ahead of time.

A fundamental Azure design principle is to scale out by adding machines or service instances based on increased demand. Scaling out can be better than purchasing more hardware that might not be in your budget. Depending on your payment plan, you don't have to pay for idle virtual machines (VMs), or need to reserve capacity in advance. A pay-as-you-go plan can be ideal for applications that need to meet planned spikes in traffic.

> **Note**
>
> Don't plan for capacity to always meet the highest level of expected demand. An inappropriate or misconfigured service can impact cost. For example, building a multiregion service when the service levels don't require high availability or georedundancy increases cost without reasonable business justification.

Choose the right resources

Right-sizing your infrastructure to meet the needs of your applications can save costs compared to the one-size-fits-all solution often employed with on-premises hardware. You can choose various options when you deploy Azure VMs to support workloads.

Each VM type has specific features and different combinations of CPU, memory, and disks. For example, B-Series VMs are ideal for workloads that don't need full CPU performance continuously, like web servers, proofs of concept, small databases, and development build environments. B-Series VMs offer a cost effective way to deploy workloads that have bursts in performance and don't need full continuous CPU performance. For a list of VM sizes and recommended uses, see Sizes for virtual machines in Azure.

Continually monitor workloads to find out if your VMs aren't optimized or have frequent unused periods. It makes sense to either downscale these VMs by using virtual machine scale sets, or shut them down. You can optimize VMs with Azure Automation, virtual machine scale sets, autoshutdown, or scripted or third-party solutions. For more information, see Automate VM optimization.

Along with choosing the right VMs, selecting the right storage type can save your organization significant monthly costs. For a list of storage data types, access tiers, storage account types, and storage redundancy options, see Select the right storage.

Use metrics to fine-tune scaling

It can be difficult to understand the relationship between metrics and capacity requirements, especially when an application is initially deployed. Provision a little extra capacity at the beginning, and then monitor and tune the *autoscaling* rules to bring capacity closer to the actual load.

Autoscaling lets you run the right amount of resources to handle your app load by adding resources or scaling out to handle increases in load such as seasonal workloads and customer-facing applications. After you configure the autoscaling rules, monitor the performance of your application over time. Use the results of this monitoring to adjust how the system scales if necessary.

Azure Monitor autoscale provides a common set of autoscaling functionality for virtual machine scale sets and Azure App Service. Scaling can be done on a schedule, or based on a runtime metric, such as CPU or memory usage. For example, you can scale out by one instance if average CPU usage is above 70%, and scale in by one instance if CPU usage falls below 50 percent.

The default autoscaling rules execute an autoscaling action at the correct time to prevent the system from reacting too quickly. For more information, see Autoscaling.

For a list of built-in metrics, see Azure Monitor autoscaling common metrics. You can also implement custom metrics by using Application Insights to monitor the performance of your live applications. Some Azure services use different scaling methods.

Scale preemptively and schedule autoscaling

Preemptive scaling based on historical data can help ensure your application performs consistently, even though your metrics haven't yet indicated the need to scale. Schedule-based rules allow you to scale when you see time patterns in your load, and you want to scale before a possible load increase or decrease occurs. For example, you can set a trigger attribute to scale out to 10 instances on weekdays, and scale in to four instances on weekends.

If you can predict the load on the application, consider using scheduled autoscaling, which adds and removes instances to meet anticipated peaks in demand. For more information, see Use Azure Monitor autoscale.

Test capacity planning

The business should communicate any fluctuation in expected load for performance testing. Load can be impacted by world events, such as political, economic, or weather changes. Load can also be affected by marketing initiatives, such as sales or promotions, or by seasonal events like holidays.

Test load variations prior to events, including unexpected load variations, to ensure that your application can scale. You should also ensure that all regions can adequately scale to support total load if one region fails.

Next steps

Performance testing

Capacity best practices

Article • 05/18/2023

The following practices help you maintain sufficient capacity to meet your application's performance efficiency needs.

Use Content Delivery Networks

Content Delivery Networks (CDNs) cache and deliver static objects from Azure Blob Storage, web applications, or any publicly accessible web server by using the closest point of presence (POP) server. CDNs can also accelerate dynamic content, which can't be cached, by using various network and routing optimizations. For more information, see What is a content delivery network on Azure.

Prepare for large-scale events

Work with your business and marketing teams to prepare for large-scale events like sports matches, sales, or marketing campaigns. You can anticipate sudden spikes in traffic caused by these events to prepare infrastructure ahead of time.

Choose the right resources

Right sizing your infrastructure to meet the needs of your applications can save costs compared to the one-size-fits-all solution often employed with on-premises hardware. Identify the needs of your application and choose the resources that best fit those needs. For more information, see Sizes for virtual machines in Azure.

Use metrics to adjust scaling policies

Autoscaling rules can use a detection mechanism based on a measured trigger attribute, such as CPU usage or queue length. These rules use an aggregated value over time, rather than instantaneous values, to trigger an autoscaling action. By default, the aggregate is an average of the values, which prevents the system from reacting too quickly or causing rapid oscillation. For more information, see Use Azure Monitor autoscale.

Preemptively scale based on trends

Preemptive scaling based on historical data can ensure your application has consistent performance, even though your metrics haven't yet indicated the need to scale. If you can predict the load on the application, consider using scheduled autoscaling, which adds and removes instances to meet anticipated peaks in demand. For more information, see Autoscaling.

Automate scale operations

Fluctuation in application traffic is expected. To ensure optimal operation conditions are maintained, such variations should be met by automated scalability operations. Autoscaling enables a platform as a service (PaaS) or infrastructure as a service (IaaS) to scale within a preconfigured range of resources. However, provisioning or deprovisioning capacity, for example adding scale units like clusters, compute instances, or deployments, is more advanced and complex. The process should be codified and automated, and the effects of adding or removing capacity should be well understood. For more information, see Repeatable infrastructure.

Monitor application health

Any change in the health state of application components can influence the capacity demands on other components. These impacts need to be fully understood, and autoscaling measures must be in place to handle the impacts. For example, if an outage in an external API is mitigated by writing messages into a retry queue, this queue needs to handle the sudden spikes in load.

Checklist - Testing for performance efficiency

Article • 05/30/2023

Is the application tested for performance, scalability, and resiliency?

Performance testing helps to maintain systems properly and fix defects before problems reach system users. It's part of the Performance Efficiency pillar in the Microsoft Azure Well-Architected Framework.

Performance testing is the superset of both load testing and stress testing. The primary goal of performance testing is to validate benchmark behavior for the application.

Load Testing validates application scalability by rapidly or gradually increasing the load on the application until it reaches a threshold.

Stress Testing is a type of negative testing that involves various activities to overload existing resources and remove components. This testing lets you understand overall resiliency and how the application responds to issues.

Use the following checklist to review your application architecture from a performance testing standpoint.

Performance testing

- **Ensure solid performance testing with shared *team* responsibility**. Successfully implementing meaningful performance tests requires many resources. It's not just a single developer or QA analyst running some tests on their local machine. Instead, performance tests need a test environment (also known as a *test bed*) that tests can be executed against without interfering with production environments and data. Performance testing requires input and commitment from developers, architects, database administrators, and network administrators.

- **Capacity planning**. When performance testing, the business must communicate any fluctuation in expected load. Load can be impacted by:
  - World events, such as political, economic, or weather changes.
  - Marketing initiatives, such as sales or promotions.
  - Seasonal events, such as holidays.

  Test variations of load prior to events, including unexpected ones, to ensure that your application can scale. Additionally, you should ensure that all regions can adequately scale to support total load, should one region fail.

- **Identify a path forward to leveraging existing tests or the creation of new tests**. There are different types of performance testing: load testing, stress testing, API testing, client-side/browser testing, and so on. It's important that you understand and articulate the different types of tests, along with their advantages and disadvantages, to the customer.

- **Perform testing in all stages in the development and deployment life cycle**. Application code, infrastructure automation, and fault tolerance should all be tested. This step ensures that the application performs as expected in every situation. You want to test early enough in the application life cycle to catch and fix errors. Errors are cheaper to repair when caught early and can be expensive or impossible to fix later. To learn more, reference Testing your application and Azure environment.

- **Avoid experiencing poor performance with testing**. Two subsets of performance testing—load testing and stress testing—can determine the upper limit and maximum point of failure, respectively, of the application's capacity. By performing these tests, you can determine the necessary infrastructure to support the anticipated workloads.

- **Plan for a load buffer to accommodate random spikes without overloading the infrastructure**. For example, if a normal system load is 100,000 requests per second, the infrastructure should support 100,000 requests at 80% of total capacity (for example, 125,000 requests per second). If you expect that the application continues to sustain 100,000 requests per second, and the current Stock Keeping Unit (SKU) introduces latency at 65,000 requests per second, you'll most likely need to upgrade your product to the next higher SKU. If there's a secondary region, you'll need to ensure that it also supports the higher SKU.

- **Test failover in multiregions**. Test the amount of time it would take for users to be rerouted to the paired region so that the region doesn't fail. Typically, a planned test failover can help determine how much time would be required to fully scale to support the redirected load.

- **Configure the environment based on testing results to sustain performance efficiency**. Scale out or scale in to handle increases and decreases in load. For example, you might know that you encounter high levels of traffic during the day and low levels on weekends. You can configure the environment to scale out for increases in load or scale in for decreases before the load actually changes.

Testing tools

- **Choose testing tools based on the type of performance testing you are attempting to execute**. There are various performance testing tools available for Azure DevOps. Some tools, like JMeter, only perform testing against endpoints and test HTTP statuses. Other tools, such as K6 and Selenium, can perform tests that also check data quality and variations. Azure Load Testing lets you create a load test by using an existing JMeter script, and monitor client-side and server-side metrics to identify performance bottlenecks. Application Insights, while not necessarily designed to test server load, can test the performance of an application within the user's browser.

- **Carry out performance profiling and load testing** during development, as part of test routines, and before final release to ensure the application performs and scales as required. This testing should occur on the same type of hardware as the production platform, and with the same types of data, quantities of data, and user load as it encounters in production.

- **Determine if it is better to use automated or manual testing**. Testing can be automated or manual. Automating tests is the best way to make sure that they're executed. Depending on how frequently tests are performed, they're typically limited in duration and scope. Manual testing is run much less frequently.

- **Cache data to improve performance, scalability, and availability**. The more data that you have, the greater the benefits of caching become. Caching typically works well with data that is immutable or that changes infrequently.

- **Decide how you'll handle local development and testing when some static content is expected to be served from a content delivery network (CDN)**. You could predeploy the content to the CDN as part of your build script. Or, use compile directives or flags to control how the application loads the resources. For example, in debug mode, the application could load static resources from a local folder. In release mode, the application would use the CDN.

- **Simulate different workloads on your application and measure application performance for each workload**. This technique is the best way to figure out what resources you need to host your application. Use performance indicators to assess whether your application is performing as expected or not.

Recommendation

Define a testing strategy. For more information, see Testing.

Next steps

Performance testing

Performance testing

Article • 05/30/2023

Performance testing helps to maintain systems properly and fix defects before problems reach system users. It helps maintain the efficiency, responsiveness, scalability, and speed of applications when compared with business requirements. When done effectively, performance testing should give you the diagnostic information necessary to eliminate bottlenecks, which lead to poor performance. A bottleneck occurs when data flow is either interrupted or stops due to insufficient capacity to handle the workload.

To avoid experiencing poor performance, commit time and resources to testing system performance. Two subsets of performance testing—load testing and stress testing—can determine the upper (close to capacity) limit and maximum (point of failure) limit, respectively, of the application's capacity. By performing these tests, you can determine the necessary infrastructure to support the anticipated workloads.

A best practice is to plan for a load buffer to accommodate random spikes without overloading the infrastructure. For example, if a normal system load is 100,000 requests per second, the infrastructure should support 100,000 requests at 80% of total capacity (that is, 125,000 requests per second). If you anticipate that the application will continue to sustain 100,000 requests per second, and the current Stock Keeping Unit (SKU) introduces latency at 65,000 requests per second, consider upgrading your product to the next higher SKU. If there's a secondary region, you must ensure that it also supports the higher SKU.

Depending on the scale of your performance test, you must plan for and maintain a testing infrastructure. You can use a cloud-based tool, such as Azure Load Testing, to abstract the infrastructure needed to run your performance tests.

Establish baselines

Baselines help to determine the current efficiency state of your application and its supporting infrastructure. Baselines can provide good insights for improvements and determine if the application is meeting business goals. Baselines can be created for any application regardless of its maturity.

First, establish performance baselines for your application. Then, establish a regular cadence for running the tests. Run the test as part of a scheduled event or part of a continuous integration (CI) build pipeline.

No matter when you establish the baseline, measure performance against that baseline during continued development. When code or infrastructure changes, the effect on performance can be actively measured.

Load testing

Load testing measures system performance as the workload increases. It identifies where and when your application breaks so you can fix the issue before shipping to production. It does so by testing system behavior under typical and heavy loads.

Load testing takes places in stages of load. These stages are usually measured by virtual users (VUs) or simulated requests, and the stages happen over given intervals. Load testing provides insights into how and when your application needs to scale to continue meeting your SLA to your customers, both internal and external. Load testing can also be useful for determining latency across distributed applications and microservices.

Consider the following key points for load testing:

- **Know the Azure service limits:** Different Azure services have *soft* and *hard* limits associated with them. The terms *soft limit* and *hard limit* describe the current, adjustable service limit (soft limit) and the maximum limit (hard limit). Understand the limits for the services you consume so that you aren't blocked if you need to exceed them. For a list of the most common Azure limits, see Azure subscription and service limits, quotas, and constraints.

  The ResourceLimits sample shows how to query the limits and quotas for commonly used resources.

- **Measure typical loads:** Knowing the typical and maximum loads on your system helps you understand when something is operating outside of its designed limits. Monitor traffic to understand application behavior.

- **Understand application behavior under various scales:** Load test your application to understand how it performs at various scales. First, test to see how the application performs under a typical load. Then, test to see how it performs under load using different scaling operations. To get more insight into how to evaluate your application as the amount of traffic sent to it increases, see Autoscale best practices.

Stress testing

Unlike load testing, which ensures that a system can handle what it's designed to handle, stress testing focuses on overloading the system until it breaks. A stress test determines how stable a system is and its ability to withstand extreme increases in load. For example, it can test the maximum number of requests from another service that a system can handle at a given time before performance is compromised and fails. Find this maximum to understand what kind of load the current environment can adequately support.

Determine the maximum demand you want to place on memory, CPU, and disk IOPS. After a stress test has been performed, you know the maximum supported load and an operational margin. It's best to choose an operational threshold so that scaling can be performed before the threshold has been reached.

Once you determine an acceptable operational margin and response time under typical loads, verify that the environment is configured adequately. To verify the configuration, make sure the SKUs that you selected are based on the desired margins. Be careful to stay as close as possible to your margins. Allocating too much can increase costs and maintenance unnecessarily. Allocating too few can result in a poor user experience.

In addition to stress testing through increased load, you can stress test by reducing resources to identify what happens when the machine runs out of memory. You can also stress test by increasing latency (for example, the database takes 10 times longer to reply, writes to storage takes 10 times longer, and so on).

Multi-region testing

A multi-region architecture can provide higher availability than deploying to a single region. If a regional outage affects the primary region, you can use Front Door to use the secondary region. This architecture can also help if an individual subsystem of the application fails.

Test the amount of time it would take for users to be rerouted to the paired region so that the region doesn't fail. To learn more about routing, see Front Door routing methods. Typically, a planned test failover can help determine how much time would be required to fully scale to support the redirected load.

Configure the environment based on testing results

Once you have performed testing and found an acceptable operational margin and response under increased levels of load, configure the environment to sustain performance efficiency. Scale out or scale in to handle increases and decreases in load. For example, you might know that you encounter high levels of traffic during the day and low levels on weekends. You can configure the environment to scale out for increases in load or scale in for decreases before the load actually changes.

For more information on autoscaling, see Design for scaling in the Performance Efficiency pillar.

> **Note**
>
> Ensure that a rule has been configured to scale the environment back down once load drops below the set thresholds. This action helps save you money.

Next steps

Testing tools

Load Testing

Article • 05/30/2023

Testing at expected peak load

Load test your application at the expected peak load to ensure there are no challenges with performance or stability when operating at full capacity. Review Changes to load test functionality in Visual Studio and cloud load testing in Azure DevOps to get an overview of the Microsoft provided load testing tools, along with a comprehensive list of third-party tools that you can use.

Azure service limits

Different Azure services have soft and hard limits associated with them. Understand the limits for the services you consume so that you aren't blocked if you need to exceed them. Review Azure subscription and service limits, quotas, and constraints to get a list of the most common Azure limits.

The ResourceLimits sample shows how to query the limits and quotas for commonly used resources.

Understanding application behavior under load

Load test your application to understand how it performs at various scales. Review Autoscale best practices to get more insight into how you can evaluate your application as the amount of traffic sent to it increases.

Measuring typical loads

Knowing the typical and maximum loads on your system helps you understand when something is operating outside of its designed limits. Monitor traffic to your application to understand user behavior.

Caching

Applications should implement a strategy that helps to ensure that the data in the cache is as up to date as possible, but can also detect and handle situations that arise when the data in the cache has become stale. Review the Cache-Aside pattern to learn how to load data on demand into a cache from a data store. Doing so can improve performance and also helps to maintain consistency between data held in the cache and data in the underlying data store.

Availability of SKUs

Certain Azure SKUs are only available in certain regions. Understand which SKUs are available in the regions you operate in so you can plan accordingly. Read about global infrastructure services .

Related sections

Performance tuning scenario: Event streaming with Azure Functions

Testing strategies

Article • 05/30/2023

There are multiple stages in the development and deployment life cycle in which tests can be performed. Application code, infrastructure automation, and fault tolerance should all be tested. Testing in various stages can ensure that the application performs as expected in every situation. You want to test early enough in the application life cycle to catch and fix errors. Errors are cheaper to repair when caught early and can be expensive or impossible to fix later.

Testing can be automated or manual. Automating tests is the best way to make sure that they're executed. Depending on how frequently tests are performed, they're typically limited in duration and scope. Manual testing is run much less frequently. For a list of tests that you should consider while developing and deploying applications, see Testing your application and Azure environment.

Identify baselines and goals for performance

Knowing where you are (baseline) and where you want to be (goal) makes it easier to plan how to get there. Established baselines and goals help you to stay on track and measure progress. Testing might also uncover a need to perform more testing on areas for which you might not have planned.

Baselines can vary based on connections or platforms used for accessing the application. It can be important to establish baselines that address the different connections, platforms, and elements such as time of day, or weekday versus weekend.

There are many types of goals when determining baselines for application performance. Some examples are the time it takes to render a page, or a desired number of transactions if your site conducts e-commerce. The following list shows some examples of questions that can help you determine goals.

What are your baselines and goals for:

- Establishing an initial connection to a service?
- An API endpoint complete response?
- Server response times?
- Latency between systems/microservices?
- Database queries?

Caching data

Caching can dramatically improve performance, scalability, and availability. The more data that you have, the greater the benefits of caching become. Caching typically works well with data that is immutable or that changes infrequently. Examples include reference information such as product and pricing information in an e-commerce application, or shared static resources that are costly to construct. Some or all of this data can be loaded into the cache at application startup to minimize demand on resources and to improve performance.

Use performance testing and usage analysis to determine whether prepopulating or on-demand loading of the cache, or a combination of both, is appropriate. The decision should be based on the volatility and usage pattern of the data. Cache utilization and performance analysis are important in applications that encounter heavy loads and must be highly scalable.

To learn more about how to use caching as a solution in testing, see Caching.

Use Azure Cache for Redis to cache data

Azure Cache for Redis is a caching service that can be accessed from any Azure application, whether the application is implemented as a cloud service, a website, or inside an Azure virtual machine. Caches can be shared by client applications that have the appropriate access key. It's a high-performance caching solution that provides availability, scalability, and security.

To learn more about using Azure Cache for Redis, see Considerations for implementing caching in Azure.

Content delivery network

Content delivery networks (CDNs) are typically used to deliver static content such as images, style sheets, documents, client-side scripts, and HTML pages. The major advantages of using a CDN are lower latency and faster delivery of content to users, regardless of their geographical location in relation to the datacenter where the application is hosted. CDNs can help to reduce load on a web application because the application doesn't have to service requests for the content that is hosted in the CDN. Using a CDN is a good way to minimize the load on your application and maximize availability and performance. Consider adopting this strategy for all of the appropriate content and resources your application uses.

Decide how to handle local development and testing when some static content is served from a CDN. For example, you could predeploy the content to the CDN as part of your build script. Alternatively, use compile directives or flags to control how the application loads the resources. For example, in debug mode, the application could load static resources from a local folder. In release mode, the application would use the CDN.

To learn more about CDNs, see Best practices for using content delivery networks (CDNs).

Benchmark testing

Benchmarking is the process of simulating different workloads on your application and measuring application performance for each workload. It's the best way to figure out what resources you need to host your application.

Use performance indicators to assess whether your application is performing as expected or not.

For workloads running on virtual machines, take into consideration VM sizes and disk sizes when benchmarking, since you might hit a particular bottleneck. The Optimize IOPS, throughput, and latency table offers further guidance.

Tools such as Azure Load Testing can help you simulate load and different usage patterns. The simulation can help you prepare for particular scenarios that are relevant to your organization or industry, for example, how a promotion or flash sale might affect an online store.

Metrics

Metrics measure trends over time. They're available for interactive analysis in the Azure portal with Azure Monitor metrics explorer. Metrics also can be added to an Azure dashboard for visualization in combination with other data and used for near-real time alerting.

Performance testing lets you see specific details on the processing capabilities of applications. You'll most likely want a monitoring tool that lets you discover proactively if the issues you find through testing are appearing in both your infrastructure and applications. Azure Monitor Metrics is a feature of Azure Monitor that collects metrics from monitored resources into a time series database.

With Azure Monitor, you can collect, analyze, and act on telemetry from your cloud and on-premises environments. It helps you understand how applications are performing and identifies issues affecting them and the resources they depend on.

For a list of Azure metrics, see Supported metrics with Azure Monitor.

Next steps

Performance monitoring

Monitor the performance of a cloud application

Article • 05/04/2023

Troubleshooting an application's performance requires monitoring and reliable investigation. Issues in performance can arise from database queries, connectivity between services, under-provisioned resources, or memory leaks in code.

Continuously monitoring services and checking the health state of current workloads is key in maintaining the overall performance of the workload. An overall monitoring strategy considers these factors:

- Scalability
- Resiliency of the infrastructure, application, and dependent services
- Application and infrastructure performance

Checklist

How are you monitoring to ensure the workload is scaling appropriately?

- ✓ Enable and capture telemetry throughout your application to build and visualize end-to-end transaction flows for the application.
- ✓ See metrics from Azure services, such as CPU and memory utilization, bandwidth information, current storage utilization, and more.
- ✓ Use resource and platform logs to get information about what events occur and under which conditions.
- ✓ For scalability, look at the metrics to determine how to provision resources dynamically and scale with demand.
- ✓ In the collected logs and metrics, look for signs that might make a system or its components suddenly become unavailable.
- ✓ Use log aggregation technology to gather information across all application components.
- ✓ Store logs and key metrics of critical components for statistical evaluation and predicting trends.
- ✓ Identify antipatterns in the code.

In this section

Follow these questions to assess the workload at a deeper level.

Assessment	Description
Are application logs and events correlated across all application components?	Correlate logs and events for subsequent interpretation. This correlation gives you visibility into end-to-end transaction flows.
Are you collecting Azure Activity Logs within the log aggregation tool?	Collect platform metrics and logs to get visibility into the health and performance of services that are part of the architecture.
Are application and resource-level logs aggregated in a single data sink, or is it possible to cross-query events at both levels?	Implement a unified solution to aggregate and query application and resource-level logs, such as Azure Log Analytics.

Azure services

The monitoring operations should utilize Azure Monitor. You can analyze data, set up alerts, get end-to-end views of your applications, and use machine learning–driven insights to identify and resolve problems quickly. Export logs and metrics to services such as Azure Log Analytics or an external service like Splunk. Also, application technologies such as Application Insights can enhance the telemetry coming out of applications.

Next section

Based on insights gained through monitoring, optimize your code. One option might be to consider other Azure services that might be more appropriate for your objectives.

Optimize

Related links

> Back to the main article

Application profiling considerations for performance monitoring

Article • 05/19/2023

Continuously monitor the application with Application Performance Monitoring (APM) technology, such as Application Insights. This technology can help you manage the performance and availability of the application, aggregating application level logs and events for subsequent interpretation.

Key points

> - Enable instrumentation and collect data using Application Insights.
> - Use distributed tracing to build and visualize end-to-end transaction flows for the application.
> - Separate logs and events of a noncritical environment from a production environment.
> - Include end-to-end transaction times for key technical functions.
> - Correlate application log events across critical system flows.

Application logs

Are application logs collected from different application environments?

Application logs support the end-to-end application lifecycle. Logging is essential in understanding how the application operates in various environments and what events occur and under which conditions.

Collect application logs and events across all application environments. Use a sufficient degree of separation and filtering to ensure noncritical environments aren't mixed with production log interpretation. Also, corresponding log entries across the application should capture a correlation ID for their respective transactions.

Are log messages captured in a structured format?

Structured format in a well-known schema can help expedite parsing and analyzing logs. Structured data can be indexed, queried, and reported without complexity.

Also, application events should be captured as a structured data type with machine-readable data points rather than unstructured string types.

Application instrumentation

Do you have detailed instrumentation in the application code?

Instrumentation of your code allows precise detection of underperforming pieces when load or stress tests are applied. It's critical to have this data available to improve and identify performance opportunities in the application code.

Use an APM such as Application Insights to continuously improve performance and usability. You need to enable Application Insights by installing an instrumentation package. The service provides extensive telemetry out of the box. You can customize what is captured for greater visibility. After it's enabled, metrics and logs related to the performance and operations are collected. View and analyze the captured data in Azure Monitor.

Distributed tracing

Events coming from different application components or different component tiers of the application should be correlated to build end-to-end transaction flows. Use distributed tracing to build and visualize flows for the application. For example, this is often achieved by using consistent correlation IDs transferred between components within a transaction.

Are application events correlated across all application components?

Event correlation between the layers of the application provides the ability to connect tracing data of the complete application stack. Once this connection is made, you can see a complete picture of where time is spent at each layer. You can then query the repositories of tracing data in correlation to a unique identifier that represents a completed transaction that has flowed through the system.

For more information, see Distributed tracing.

Critical targets

Is it possible to evaluate critical application performance targets and nonfunctional requirements (NFRs)?

Application level metrics should include end-to-end transaction times of key technical functions, such as database queries, response times for external API calls, failure rates of processing steps, and more.

Is the end-to-end performance of critical system flows monitored?

It should be possible to correlate application log events across critical system flows, such as user sign-in, to fully assess the health of key scenarios in the context of targets and NFRs.

Cost optimization

If you're using Application Insights to collect instrumentation data, there are cost considerations. For more information, see Manage usage and costs for Application Insights.

Next

Monitor infrastructure

Related links

- Distributed tracing
- Application Insights
- Azure Monitor

> Back to the main article

Analyze infrastructure metrics and logs

Article • 05/04/2023

Performance issues can occur because of interaction between the application and other services in the architecture. For example, issues in database queries, connectivity between services, and under-provisioned resources are all common causes for inefficiencies.

The practice of continuous monitoring must include analysis of platform metrics and logs to get visibility into the health and performance of services that are part of the architecture.

Key points

> - View platform metrics to get visibility into the health and performance of Azure services.
> - Use log data to get visibility into the operations and events of the management plane.
> - Track events from internal dependencies.
> - Check the health of external dependencies, such as an API service.

Platform metrics

Metrics are numerical values that are collected at regular intervals and describe some aspect of a system at a particular time. View the platform metrics that are generated by the services used in the architecture. Each Azure service has a set of metrics that's unique to the functionality of the resource. These metrics give you visibility into their health and performance. There's no added configuration for Azure resources. You can also define custom metrics for an Azure service using the custom metrics API.

Azure Monitor Metrics is a feature of Azure Monitor that collects numeric data from monitored resources into a time series database. To learn more about Azure Monitor Metrics, see What can you do with Azure Monitor Metrics?.

If your application is running in Azure Virtual Machines, configure Azure Diagnostics extension to send guest OS performance metrics to Azure Monitor. Guest OS metrics include performance counters that track guest CPU percentage or memory usage, both of which are frequently used for autoscaling or alerting.

For more information, see Supported metrics with Azure Monitor.

Also, use technology-specific tools for the services used in the architecture. For example, use network traffic capturing tools, such as Azure Network Watcher.

One of the challenges to metric data is that it often has limited information to provide context for collected values. Azure Monitor addresses this challenge with multi-dimensional metrics. These metrics are name-value pairs that carry more data to describe the metric value. To learn about multi-dimensional metrics and an example for network throughput, see multi-dimensional metrics.

Platform logs

Azure provides various operational logs from the platform and the resources. These logs provide insight into what events occurred, what changes were made to the resource, and more. These logs are useful in tracking operations. For example, you can track scaling events to check if autoscaling is working as expected.

Azure Monitor Logs can store various different data types each with their own structure. You can also perform complex analysis on logs data using log queries, which can't be used for analysis of metrics data. Azure Monitor Logs is capable of supporting near real-time scenarios, making them useful for alerting and fast detection of issues. To learn more about Azure Monitor Logs, see What can you do with Azure Monitor Logs?.

Are you collecting Azure activity logs within the log aggregation tool?

Azure activity logs provide audit information about when an Azure resource is modified, such as when a virtual machine is started or stopped. This information is useful for the interpretation and troubleshooting of issues. It provides transparency around configuration changes that can be mapped to adverse performance events.

Are logs available for critical internal dependencies?

To build a robust application health model, ensure there's visibility into the operational state of critical internal dependencies.

In Azure Monitor, enable Azure resource logs so that you have visibility into operations that were done within an Azure resource. Similar to platform metrics, resource logs vary by the Azure service and resource type.

For example, for services such as a shared network virtual appliance (NVA) or Express Route connection, monitor the network performance. Azure monitor can also help diagnose networking related issues. You can trigger a packet capture, diagnose routing issues, analyze network security group flow logs, and gain visibility and control over your Azure network.

Here are some tools you can use:

- Network performance monitor
- Service connectivity monitor
- ExpressRoute monitor

Also, data from network traffic capturing tools, such as Network Watcher, can be helpful.

Are critical external dependencies monitored?

Monitor critical external dependencies, such as an API service, to ensure operational visibility of performance. For example, a probe can be used to measure the latency of an external API.

Cost optimization for monitoring

Azure Monitor billing model is based on consumption. Azure creates metered instances that track usage to calculate your bill. Pricing depends on the metrics, alerting, notifications, Log Analytics, and Application Insights.

For information about usage and estimated costs, see Monitoring usage and estimated costs in Azure Monitor.

You can also use the pricing calculator to determine your pricing. The pricing calculator helps you estimate your likely costs based on your expected use.

Next

Data analysis considerations

Related links

- Supported metrics with Azure Monitor
- Network performance monitor
- Azure Network Watcher

> Back to the main article

Performance data integration

Article • 05/19/2023

Performance testing and investigation should be based on data captured from repeatable processes. To understand how an application's performance is affected by code and infrastructure changes, retain data for analysis. Also, it's important to measure how performance has changed *over time*, not just compared to the last measurement taken.

This article describes some considerations and tools you can use to aggregate data for troubleshooting and analyzing performance trends.

Key points

> - Analyze performance data holistically to detect fault types, bottleneck regressions, and health states.
> - Use log aggregation technologies to consolidate data into a single workspace and analyze using a sophisticated query language.
> - Retain data in a time-series database to predict performance issues before they occur.
> - Balance the retention policy and service pricing plans with the cost expectation of the organization.

Data interpretation

The overall performance can be impacted by both application-level issues and resource-level failures. It's vital that all data is correlated and evaluated together to optimize the detection of issues and troubleshooting of detected issues. This approach helps distinguish between transient and nontransient faults.

Use a holistic approach to quantify what *healthy* and *unhealthy* states represent across all application components. We recommend that a *traffic light* model is used to indicate a healthy state. For example, use a green light to show that key nonfunctional requirements and targets are fully satisfied and resources are optimally used. For example, a healthy state can mean that 95% of requests are processed in <= 500 ms with AKS node utilization at x%, and so on. Also, an Application Map can help spot performance bottlenecks or failure hotspots across components of a distributed application.

Analyze long-term operational data to get historical context and detect if there have been any regressions. For example, check the average response times to see if they've been slowly increasing over time and getting closer to the maximum target.

Aggregated view

Log aggregation technologies should be used to collate logs and metrics across all application components, including infrastructural components for later evaluation.

Resources can include Azure infrastructure as a service (IaaS) and platform as a service (PaaS) services and third-party appliances, such as firewalls or antimalware solutions used in the application. For example, if Azure Event Hubs is used, the diagnostic settings should be configured to push logs and metrics to the data sink.

Azure Monitor can collect and organize log and performance data from monitored resources. Data is consolidated into an Azure Log Analytics workspace so they can be analyzed together using a sophisticated query language that can quickly analyze millions of records. Splunk is another popular choice.

How is aggregated monitoring enforced?

All application resources should be configured to route diagnostic logs and metrics to the chosen log aggregation technology. Use Azure Policy to ensure the consistent use of diagnostic settings across the application and to enforce the desired configuration for each Azure service.

Long-term data

Store long-term operational data to understand the history of application performance. This data store is important for analyzing performance trends and regressions.

Are long-term trends analyzed to predict performance issues before they occur?

It's often helpful to store such data in a time-series database (TSDB) and then view the data from an operational dashboard. An Azure Data Explorer cluster is a powerful TSDB that can store any schema of data, including performance test metrics. Grafana, an open-source platform for observability dashboards, can then be used to query your Azure Data Explorer cluster to view performance trends in your application.

Have retention times been defined for logs and metrics, with housekeeping mechanisms configured?

Clear retention times should be defined to allow for suitable historic analysis and to control storage costs. Suitable housekeeping tasks should also be used to archive data to cheaper storage or aggregate data for long-term trend analysis.

Cost optimization

While correlating data is recommended, there are cost implications to storing long-term data. For example, Azure Monitor is capable of collecting, indexing, and storing massive amounts of data in a Log Analytics workspace. The cost of a Log Analytics workspace is based on amount of storage, retention period, and the plan. For more information about how you can balance cost, see Manage usage and costs with Azure Monitor Logs.

If you're using Application Insights to collect instrumentation data, there are cost considerations. For more information, see Manage usage and costs for Application Insights.

Next

Scalability and reliability considerations

Related links

- Application Map
- Azure Policy
- Azure Data Explorer cluster
- Manage usage and costs with Azure Monitor Logs
- Manage usage and costs for Application Insights

> Back to the main article

Monitor performance for scalability and reliability

Article • 05/19/2023

The overall monitoring strategy should take into consideration scalability and reliability of the infrastructure, application, and dependent services.

Scalability

A goal of analyzing metrics is to determine the thresholds for scale up, scale out, scale in, and scale down. The ability to scale dynamically is one of the biggest values of moving to the cloud.

- Know the minimum number of instances that should run at any given time.
- Determine the best metrics for your solution on which to base your auto scaling rules.
- Configure the auto scaling rules for those services that include it.
- Create alert rules for the services that can be scaled manually.

For scalability, look at the metrics to determine how to provision resources dynamically and scale with demand.

Reliability

Monitor your application for early warning signs that might require proactive intervention. Tools that assess the overall health of the application and its dependencies help you to recognize quickly when a system or its components suddenly become unavailable. Use them to implement an early warning system.

Monitoring systems should capture comprehensive details so that applications can be restored efficiently. Then, if necessary, designers and developers can modify the system to prevent the situation from recurring.

The raw data for monitoring can come from various sources, including:

- Application logs, such as those produced by Application Insights.
- Operating system performance metrics collected by Azure monitoring agents.
- Azure resources, including metrics collected by Azure Monitor.
- Azure Service Health, which offers a dashboard to help you track active events.
- Azure AD logs built into the Azure platform.

To learn more, see Monitoring health for reliability.

Next section

Based on insights gained through monitoring, optimize your code. One option might be to consider other Azure services that might be more appropriate for your objectives.

Optimize

Related links

> Back to the main article

Sustain performance efficiency over time

Article • 05/19/2023

Performance is an indication of the responsiveness of a system to execute any action within a given time interval. As workloads and requirements change over time, you need to continuously ensure that systems don't start to under-perform. This article explains some steps you can take to help make sure your systems are continuously running at an optimum level.

Optimize autoscaling

Autoscaling is the process of dynamically allocating resources to match performance requirements. As the volume of work grows, an application may need more resources to maintain the desired performance levels and satisfy service-level agreements (SLAs). As demand slackens, and the extra resources are no longer needed, they can be deallocated to minimize costs.

To prevent a system from attempting to scale out excessively, consider limiting the maximum number of instances that can be automatically added. Doing so helps you avoid the costs associated with running potentially many thousands of instances. Most autoscaling mechanisms allow you to specify the minimum and maximum number of instances for a rule. Also consider gracefully degrading the functionality that the system provides if the maximum number of instances have been deployed and the system is still overloaded.

But autoscaling might not always be the most appropriate mechanism to handle a sudden burst in workload. It takes time to provision and start new instances of a service or add resources to a system, and the peak demand may have passed by the time these extra resources have been made available. In this scenario, it may be better to throttle the service. For more information, see Throttling pattern.

If you do need the capacity to process all requests when the volume fluctuates rapidly, you can use an aggressive autoscaling strategy. Aggressive autoscaling does increase costs, but it adds the extra instances you need more quickly. You can also use a scheduled policy that starts a sufficient number of instances to meet the maximum load before that load is expected.

Separate out critical workload

Some data is accessed more frequently than other data. If you store data in its own partition depending on its usage pattern, your system can run more efficiently. Operations that affect more than one partition can run in parallel.

Partitioning data can improve the availability of applications by ensuring that the entire dataset doesn't constitute a single point of failure and that subsets of the dataset can be managed independently. For this reason, you might want to separate out critical workload from noncritical. If one instance fails, only the data in that partition is unavailable. Operations on other partitions can continue. For managed PaaS data stores, this consideration is less relevant, because these services are designed with built-in redundancy.

To learn more, see Data partitioning.

Right-size your resources

Changes to the resources that support a workload may affect the architecture of the workload. When this situation happens, other considerations are required to minimize the effect on end users and business functions. One of these considerations is right-sizing, which is about controlling cost by continuously monitoring and adjusting size of your instances to meet needs. Cost shouldn't necessarily be the main decision-making factor. Choosing the least expensive option could expose the workload to performance and availability risks.

For example, when you consider pricing and sizing resources hosted in Azure, right-sizing virtual machines (VMs) is best practice. Choosing the right storage type for data can save your organization several thousands of dollars every month. Microsoft offers many options, and each VM type has specific features and different combinations of CPU, memory, and disks.

To learn more about right sizing best practices with VMs, see Best practice: Right-size VMs.

For accountability purposes

Identify right-size opportunities by reviewing your current resource utilization and performance requirements across the environment. Then, modify each resource to use the smallest instance or SKU that can support the performance requirements of each resource.

For other best practices, see Best practices by team and accountability.

For operational cost management purposes

Review your environment's current resource utilization and performance requirements. Then, identify resources that have remained underutilized (generally more than 90 days). Also, right-size provisioned SKUs by modifying underutilized resources to use the smallest instance or SKU that can support the performance requirements of each resource. Finally, right-size redundancy. If the resource doesn't require a high degree of redundancy, remove geo-redundant storage.

For other best practices, see Operational cost management best practices.

Remove antipatterns

A performance antipattern is a common practice that is likely to cause scalability problems when an application is under pressure. Some of the following common antipatterns can occur when a system offloads too much processing to a data store:

- Moving resource-intensive tasks onto background threads.
- Continually sending many small network requests.
- Failing to cache data.

Antipatterns can cause decreased response time, high latency, slow I/O calls, and other performance issues.

Many factors can cause an antipattern to occur. Sometimes an application inherits a design that worked on-premises, but doesn't scale in the cloud. Or, an application might start with a clean design, but as new features are added, one or more of these antipatterns can appear.

Removing antipatterns can improve performance efficiency. But removal isn't a straight-forward task, because sometimes the problem only manifests under certain circumstances. Instrumentation and logging are key to finding the root cause, but you also have to know what to look for. To learn more about common antipatterns and how to identify and fix them, see Catalog of antipatterns.

Partition data to optimize performance

Article • 05/19/2023

Data partitioning can optimize performance, improve scalability, and reduce contention by lowering the taxation of database operations. It can also provide a mechanism for dividing data by usage pattern. For example, you can archive older data in cheaper data storage. Data partitioning involves conversations and planning between developers and database administrators.

For more reasons why you might want to partition, see Why partition data?

Determine acceptable performance optimization

There's almost no limit to how much an application can be performance tuned. How do you know when you have tuned an application enough? To find your limit, use the 80/20 rule. Generally, 80% of the application can be optimized by focusing on just 20%. A company typically sees a diminishing return on further optimization after 20%. The question you should answer is how much of the remaining 80% of the application is worth optimizing for the business. For example, how much does optimizing the remaining 80% help the business reach its goals of customer acquisition, retention, sales, etc.? The business must determine its own realistic definition of "acceptable."

Types of partitioning

You can partition to hold a specific subset of the data, like all the orders for a set of customers. You can hold a subset of fields that are divided according to their pattern of use, like frequently accessed fields versus less frequently accessed fields. Or you can aggregate data according to how it's used by each bounded context in the system, like how an e-commerce system might store product inventory and invoice data in two different partitions.

To learn more about the main types of partitioning, see Horizontal, vertical, and functional data partitioning.

Strategies for data partitioning

Partitioning adds complexity to the design and development of your system. Consider partitioning as a fundamental part of system design even if the system initially contains only a single partition. If you address partitioning as an afterthought, it's more challenging because you already have a live system to maintain:

- Data access logic needs to be modified.
- Large quantities of existing data might need to be migrated to distribute it across partitions.
- Users expect to be able to continue using the system during the migration.

Different strategies are used to partition data in various Azure data stores to help improve performance. To learn more, see the following data storage links details.

- Partitioning Azure SQL Databases
- Partitioning Azure table storage
- Partitioning Azure blob storage
- Partitioning Azure storage queues
- Partitioning Azure Service Bus
- Partitioning Azure Cosmos DB
- Partitioning Azure Search
- Partitioning Azure Cache for Redis
- Partitioning Azure Service Fabric
- Partitioning Azure Event Hubs

To learn about these strategies, see Application design considerations.

Next steps

Sustain

Cache data for performance optimization

Article • 05/19/2023

Caching is a strategy where you store a copy of the data in front of the main data store. Advantages of caching include faster response times and the ability to serve data quickly, which can improve user experience. The cache store is typically located closer to the consuming client than the main store.

Caching is most effective when a client instance repeatedly reads the same data, especially if all the following conditions apply to the original data store:

- It remains relatively static.
- It's slow compared to the speed of the cache.
- It's subject to a high level of contention.
- It's far away when network latency can cause access to be slow.

Caching can dramatically improve performance, scalability, and availability. The more data that you have and the larger the number of users that need to access this data, the greater the benefits of caching become. You get this benefit because caching reduces the latency and contention that's associated with handling large volumes of concurrent requests in the original data store.

Incorporating appropriate caching can also help reduce latency by eliminating repetitive calls to microservices, APIs, and data repositories. The key to using a cache effectively lies in determining the most appropriate data to cache and caching it at the appropriate time. Data can be added to the cache on demand the first time it's retrieved by an application. Using this method means that the application needs to fetch the data only once from the data store, and it means that subsequent access can be satisfied by using the cache. To learn more, see Determine how to cache data effectively.

For details, see Caching.

Azure Cache for Redis

Azure Cache for Redis improves the performance and scalability of an application. It processes large volumes of application requests by keeping frequently accessed data in the server memory that can be written to and read from quickly. Based on the Redis software, Azure Cache for Redis brings critical low-latency and high-throughput data storage to modern applications.

Azure Cache for Redis also improves application performance by supporting common application architecture patterns. Some of the most common patterns include data cache and content cache. For the most common patterns and their descriptions, see Common application architecture patterns.

Azure Content Delivery Network (CDN)

A content delivery network (CDN) is a distributed network of servers that can efficiently deliver web content to users. A CDN stores cached content on edge servers in point-of-presence (POP) locations that are close to end users to minimize latency. To learn more about CDN, see What is a content delivery network on Azure?

Next

Partition

Performance efficiency patterns

Article • 05/26/2023

Performance efficiency is the ability of your workload to meet the demands placed on it by users in an efficient manner. You need to anticipate increases in load to meet business requirements. To achieve performance efficiency, it's important to consider how your application scales and to implement platform as a service (PaaS) offerings that have built-in scaling operations.

Performance efficiency depends on the system's ability to handle load increases without impacting performance, or for available resources to be readily increased. Performance efficiency concerns not just compute instances, but other elements such as data storage, messaging infrastructure, and application architecture.

The following table summarizes architectural design patterns that relate to performance efficiency.

Pattern	Summary
Cache-aside	Load data on demand into a cache from a data store.
Choreography	Have each component of the system participate in the decision-making process about the workflow of a business transaction, instead of relying on a central point of control.
Command query responsibility segregation (CQRS)	Segregate operations that read data from operations that update data by using separate interfaces.
Event sourcing	Use an append-only store to record the full series of events that describe actions taken on data in a domain.
Deployment stamps	Deploy multiple independent copies of application components, including data stores.
Geodes	Deploy backend services into a set of geographical nodes, each of which can service any client request in any region.
Index table	Create indexes over the fields in data stores that are frequently referenced by queries.
Materialized view	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for a required query.
Priority queue	Prioritize requests sent to services so that requests with a higher priority are received and processed more quickly than requests with a lower priority.

Pattern	Summary
Queue-based load leveling	Use a queue that acts as a buffer between a task and a service that it invokes in order to smooth intermittent heavy loads.
Sharding	Divide a data store into a set of horizontal partitions or shards.
Static content hosting	Deploy static content to a cloud-based storage service that can deliver the assets directly to the client.
Throttling	Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service.

Performance efficiency checklist

Article • 05/19/2023

Performance efficiency, one of the pillars of the Microsoft Azure Well-Architected Framework, is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. Use this checklist to review your application architecture from a performance efficiency standpoint.

Application design

- **Design for scaling**. Scaling allows applications to react to variable load by increasing and decreasing the number of roles, queues, and other services they use. The application must be designed with scaling in mind.

  The application and the services it uses must be stateless, so requests can be routed to any instance and adding or removing specific instances doesn't adversely affect users. Implement configuration, autodetection, or load balancing, so as services are added or removed, the application can do the necessary routing.

  For example, a web application might use a set of queues in a round-robin approach to route requests to background services running in worker roles. The web application must be able to detect changes in the number of queues to successfully route requests and balance the load on the application.

  You can scale outbound connectivity to the internet with Azure NAT Gateway. Azure NAT Gateway provides a scalable, reliable, and secure way to connect outbound traffic to the internet, and helps prevent connection failures caused by SNAT exhaustion.

- **Partition the workload**. Design parts of the process to be discrete and decomposable. Minimize the size of each part, and follow the usual rules for separation of concerns and the single responsibility principle. These practices allow the component parts to be distributed to maximize use of each compute unit, such as a role or database server.

  Partitioning the workload also makes it easier to scale the application by adding instances of specific resources. For complex domains, consider adopting a microservices architecture.

- **Scale as a unit**. Plan for more resources to accommodate growth. For each resource, know the upper scaling limits, and use sharding or decomposition to go beyond these limits. Scaling as a unit also makes operations less prone to negative impact from resource limitations in other parts of the overall system.

To make scale-out operations easier, determine the scale units for the system in well-defined sets of resources. For example, adding *x* number of web and worker roles might require *y* added queues and *z* storage accounts to handle the added workload. So a scale unit could consist of *x* web and worker roles, *y* queues, and *z* storage accounts.

Design the application to easily scale by adding one or more scale units. Consider using the deployment stamps pattern to deploy scale units.

- **Take advantage of platform autoscaling**. If the hosting platform, such as Azure, supports autoscaling, prefer it to custom or third-party mechanisms unless the built-in mechanism can't fulfill your requirements. Schedule scaling rules where possible to ensure resources are available without a startup delay. Add reactive autoscaling to the rules where appropriate to cope with unexpected changes in demand. For more information, see Autoscaling guidance.

> ⓘ Note
>
> You can use autoscaling operations and add custom counters to rules in the older Azure Classic deployment model. For more information, see **Increase a VM-family vCPU quota for the Classic deployment model**.

- **Avoid client affinity**. Where possible, ensure that the application doesn't require affinity. Requests can then be routed to any instance, and the number of instances is irrelevant. You also avoid the overhead of storing, retrieving, and maintaining state information for each user.

- **Offload CPU-intensive and I/O-intensive tasks as background tasks**. If a request to a service is expected to take a long time to run or consume many resources, offload the processing for this request to a separate task. Depending on hosting platform, use worker roles or background jobs to run these tasks. This strategy enables the service to continue receiving requests and remain responsive. For more information, see Background jobs guidance.

- **Distribute the workload for background tasks**. If there are many background tasks, or the tasks require substantial time or resources, spread the work across multiple compute units, such as worker roles or background jobs. For one possible solution, see the Competing consumers pattern.

- **Move toward a *shared-nothing* architecture**. A shared-nothing architecture uses independent, self-sufficient nodes that have no single point of contention like shared services or storage. In theory, such a system can scale almost indefinitely.

  While a fully shared-nothing approach isn't practical for most applications, it might provide opportunities to design for better performance. For example, avoiding the use of server-side session state, client affinity, and data partitioning are good examples of moving toward a shared-nothing architecture.

Data management

- **Use data partitioning**. Divide the data across multiple databases and database servers. You can design the application to use data storage services that provide partitioning transparently, such as Azure SQL Database elastic pools and Azure Table Storage. This approach can help maximize performance and allow easier scaling. Benefits of partitioning include better query performance and availability, simpler scalability, and easier management.

  Different partitioning techniques include *horizontal*, *vertical*, and *functional*. You can combine these techniques to achieve maximum benefits. Match different data store types to different data types, and choose data store types that are optimized for specific types of data. For example, you can use table storage, a document database, or a column-family data store instead of, or along with, a relational database. For more information, see Data partitioning guidance.

- **Design for eventual consistency**. Eventual consistency improves scalability by reducing or removing the time needed to synchronize related data partitioned across multiple stores. The cost is that data isn't always consistent when it's read, and some write operations might cause conflicts. Eventual consistency is ideal for situations where the same data is read frequently but written infrequently. For more information, see the Data consistency primer.

- **Reduce chatty interactions between components and services**. Avoid designing interactions in which an application is required to make multiple calls to a service, each of which returns a small amount of data. Instead use a single call that can return all the data. For example, use stored procedures in databases to encapsulate complex logic, reduce the number of round trips, and help prevent resource locking.

  Where possible, combine several related operations into a single request when the call is to a service or component that has noticeable latency. This practice makes it easier to monitor performance and optimize complex operations.

- **Use queues to level the load for high velocity data writes**. Surges in demand can overwhelm a service and cause escalating failures. The queue-based load leveling pattern can help prevent this situation. This pattern uses a queue as a buffer between a task and a service that it invokes. The queue can smooth intermittent heavy loads that might otherwise cause the service to fail or the task to time out.

- **Minimize the load on the data store**. The data store is a costly resource, is often a processing bottleneck, and isn't easy to scale out. It's easier to scale out the application than the data store, so you should attempt to do as much as possible of the compute-intensive processing within the application.

  Where possible, remove logic like processing XML documents or JSON objects from the data store, and do the processing within the application. For example, avoid passing XML to the database, other than as an opaque string for storage. Serialize or deserialize the XML within the application layer and pass it in a form that's native to the data store.

- **Minimize the volume of data retrieved**. Retrieve only the data you require by specifying columns and using criteria to select rows. Use table value parameters and the appropriate isolation level. Use mechanisms such as entity tags to avoid retrieving data unnecessarily.

- **Aggressively use caching**. Use caching wherever possible to reduce the load on resources and services that generate or deliver data. Caching is best suited to data that's relatively static, or that requires considerable processing to get. Where appropriate, caching should occur in each layer of the application, including data access and user interface generation. For more information, see Caching guidance.

- **Handle data growth and retention**. The amount of data stored by an application grows over time. This growth increases storage costs and data access latency, affecting application throughput and performance. If possible, periodically archive some of the old data that's no longer accessed. Or, move rarely accessed data into long-term storage that's more cost effective, even if access latency is higher.

- **Optimize Data Transfer Objects (DTOs) by using an efficient binary format**. DTOs are passed between the layers of an application many times. Minimizing their size reduces the load on resources and the network. Balance the savings with the overhead of converting the data to the required format in each location where it's used. To encourage easy reuse of a component, adopt a format that has the maximum interoperability.

- **Set cache control**. To minimize processing load, design and configure the application to use output caching or fragment caching where possible.

- **Enable client side caching**. Web applications should enable cache settings on content that can be cached, which are often disabled by default. Configure the server to deliver the appropriate cache control headers to enable caching of content on proxy servers and clients.

- **Use Azure Blob Storage and the Azure Content Delivery Network to reduce the load on the application**. Consider storing static or relatively static public content, such as images, resources, scripts, and style sheets, in blob storage. This approach relieves the application from the load of dynamically generating this content for each request.

  Also consider using the Content Delivery Network to cache this content and deliver it to clients. Using the Content Delivery Network can improve performance at the client because the content is delivered from the geographically closest datacenter that contains a Content Delivery Network cache. For more information, see Best practices for using content delivery networks.

- **Optimize and tune SQL queries and indexes**. Some T-SQL statements or constructs might have an adverse effect on performance that can be reduced by optimizing the code in a stored procedure. For example, avoid converting **datetime** types to **varchar** before comparing with a **datetime** literal value. Use date/time comparison functions instead.

  Lack of appropriate indexes can also slow query execution. If you use an object/relational mapping framework, understand how it works and how it might affect performance of the data access layer. For more information, see Query tuning.

- **Denormalize data**. Data normalization helps to avoid duplication and inconsistency. However, data normalization imposes an overhead that can affect performance. Overhead includes maintaining multiple indexes, checking for referential integrity, performing multiple accesses of small chunks of data, and joining tables to reassemble the data. Consider whether some added storage volume and duplication is acceptable to reduce the load on the data store.

  Also consider if the application itself, which is typically easier to scale, can be relied on to take over tasks such as managing referential integrity to reduce the load on the data store. For more information, see Data partitioning guidance.

Implementation

- **Avoid performance antipatterns**. Review the performance antipatterns for cloud applications for common practices that often cause performance problems when applications are under pressure.

- **Use asynchronous calls**. If possible, use asynchronous code to access resources or services that might be limited by I/O or network bandwidth or have a noticeable latency. Asynchronous calls avoid locking the calling thread.

- **Avoid locking resources, and use an optimistic approach instead**. Locking access to resources such as storage or other services that have noticeable latency is a primary cause of poor performance. Always use optimistic approaches to managing concurrent operations like writing to storage. Use storage layer features to manage conflicts. In distributed applications, data might be only eventually consistent.

- **Compress highly compressible data over high latency, low bandwidth networks**. In most web applications, HTTP responses to client requests are the largest volume of data generated by the application and passed over the network. HTTP compression can reduce this volume, especially for static content.

  Compressing dynamic content can reduce cost and load on the network, but applies a slightly higher load on the server. In more generalized environments, data compression can reduce the volume of data transmitted and minimize transfer time and costs, but the compression and decompression processes incur overhead.

  > ⓘ **Note**
  >
  > Compression should be used only when there's a demonstrable gain in performance. Other serialization methods like JSON or binary encodings might reduce payload size and have less impact on performance, whereas XML is likely to increase it.

- **Minimize the time that connections and resources are in use**. Maintain connections and resources only for as long as you need to use them. Open connections as late as possible, and allow them to be returned to the connection pool as soon as possible. Acquire resources as late as possible, and dispose of them as soon as possible.

- **Minimize the number of connections required**. Service connections absorb resources. Limit the number of required service connections, and ensure that existing connections are reused whenever possible. For example, after authentication, use impersonation where appropriate to run code as a specific identity. Impersonation helps make the best use of the connection pool by reusing connections.

> ⓘ **Note**
>
> APIs for some services automatically reuse connections, if service-specific guidelines are followed. Understand the conditions that enable connection reuse for each service that your application uses.

- **Send requests in batches to optimize network use**. For example, send and read messages in batches when you access a queue, and do multiple reads or writes as a batch when you access storage or a cache. Batching can help to maximize efficiency of the services and data stores by reducing the number of calls across the network.

- **Avoid requirements to store server-side session state**. Server-side session state management typically requires client affinity, or routing each request to the same server instance, which affects the system's ability to scale. Where possible, design clients to be stateless with respect to the servers that they use. However, if the application must maintain session state, store sensitive data or large volumes of per-client data in a distributed server-side cache that all instances of the application can access.

- **Optimize table storage schemas**. When using table stores that require the table and column names to be passed and processed with every query, such as Table Storage, consider using shorter names to reduce overhead. However, don't sacrifice readability or manageability by using overly compact names.

- **Create resource dependencies during deployment or at application startup**. Avoid repeated calls to methods that test the existence of a resource and then create the resource if it doesn't exist. Methods such as `CloudTable.CreateIfNotExists` and `CloudQueue.CreateIfNotExists` in the Azure Storage client library follow this pattern. These methods can impose considerable overhead if they're invoked before each access to a storage table or storage queue.

  Instead, create the required resources when the application is deployed, or when it first starts. A single call to `CreateIfNotExists` for each resource in the startup code for a web or worker role is acceptable. However, be sure to handle exceptions that might arise if your code attempts to access a resource that doesn't exist. In these situations, you should log the exception, and possibly alert an operator that a resource is missing.

Under some circumstances, it might be appropriate to create the missing resource as part of the exception handling code. Use this approach with caution, because the nonexistence of the resource might indicate a programming error, a misspelled resource name, or some other infrastructure-level issue.

- **Use lightweight frameworks**. Carefully choose the APIs and frameworks you use to minimize resource usage, execution time, and overall load on the application. For example, using a web API to handle service requests can reduce the application footprint and increase execution speed. But a web API might not be suitable for advanced scenarios where the added capabilities of Windows Communication Foundation are required.

- **Minimize the number of service accounts**. For example, use a specific account to access resources or services that impose a limit on connections or perform better when fewer connections are maintained. This approach is common for services such as databases, but can affect accurate auditing due to the impersonation of the original user.

- **Carry out performance profiling and load testing** during development, as part of test routines, and before final release to ensure the application performs and scales as required. This testing should occur on the same type of hardware as the production platform. Use the same types and quantities of data and user load as in production. For more information, see Test the performance of a cloud service.

Next steps

Tradeoffs for performance efficiency

Tradeoffs for performance efficiency

Article • 05/26/2023

As you design a workload, consider tradeoffs between performance efficiency and other aspects of the design, such as cost optimization, operational excellence, reliability, and security.

Performance efficiency vs. cost optimization

Cost can increase as a result of increasing performance. When you optimize for performance, consider the following factors to manage costs:

- Avoid estimating workload costs at consistently high utilization. Smooth out the peaks to get a consistent flow of compute and data. Consumption-based pricing is more expensive than the equivalent provisioned pricing. Ideally, use manual scaling and autoscaling to find the right balance. Scaling up is usually more expensive than scaling out.
- Costs increase directly with number of regions. Locating resources in cheaper regions doesn't negate the cost of network ingress and egress, or degraded application performance because of increased latency.
- Every render cycle of a payload consumes both compute and memory. You can use caching to reduce load on servers, and save with precanned storage and bandwidth costs. The savings can be dramatic, especially for static content services.

  While caching can reduce cost, there are some performance tradeoffs. For example, Azure Traffic Manager pricing is based on the number of Domain Name Service (DNS) queries that reach the service. You can reduce that number through caching, and configure how often the cache is refreshed. But relying on a cache that isn't frequently updated causes longer user failover times if an endpoint is unavailable.
- Using dedicated resources to batch process long running jobs increases the cost. You can lower costs by provisioning Azure Spot Virtual Machines, but be prepared for the job to be interrupted every time Azure evicts the virtual machine (VM).

For other cost considerations, see the Cost optimization pillar.

Performance efficiency vs. operational excellence

As you determine how to design your workload to meet the demands placed on it by users in an efficient manner, consider operations. Operations processes keep an application running in production. To achieve operational excellence with these processes, use the following guidelines:

- Ensure that deployments remain reliable and predictable.
- Automate deployments to reduce the chance of human error.
- Make the deployment process fast and routine, so it doesn't slow down the release of new features or bug fixes.
- Be able to quickly roll back or roll forward if an update has problems.

For other operational considerations, see the Operational excellence pillar.

Automate performance testing

Automated performance testing is an operational process that can help to identify performance issues early. A serious performance issue can impact a deployment as severely as a bug in the code. Automated functional tests can prevent application bugs, but they might not detect performance problems.

Define acceptable performance goals for metrics such as latency, load times, and resource usage. Include automated performance tests in your release pipeline to make sure the application meets those goals.

Do fast builds

Another operational excellence process is making sure that your product is in a deployable state through a fast build process. Builds provide crucial information about the status of your application.

The following practices promote faster builds:

- Select the right size of VMs.
- Ensure that the build server is located near the source and target locations to considerably reduce build duration.
- Scale out build servers.
- Optimize the build.

For more information, see Performance considerations for your deployment infrastructure.

Monitor performance

As you consider performance improvements, monitoring should be done to verify that your application is running correctly. Monitoring should include the application, platform, and networking. For more information, see Monitor operations of cloud applications.

Performance efficiency vs. reliability

Acknowledge up front that failures happen. Instead of trying to completely prevent failures, the goal is to minimize the effects of a single failing component.

Reliable applications are *resilient* and *highly available* (HA). Resiliency allows systems to recover gracefully from failures and continue to function with minimal downtime and data loss before full recovery. HA systems run as designed in a healthy state with no significant downtime. Maintaining reliability lets you maintain performance efficiency.

For reliability, consider the following guidelines:

- Use the circuit breaker pattern to provide stability and minimize performance impact while the system recovers from a failure.
- Segregate read and write interfaces by using the command query responsibility segregation (CQRS) pattern to achieve the scale and performance needed for your solution.
- Try to achieve higher availability by adopting an *eventual consistency* model. For more information about selecting the correct data store, see Use the best data store for your data.
- If your application requires more storage accounts than are currently available in your subscription, create a new subscription with more storage accounts. For more information, see Scalability and performance targets.
- Avoid scaling up and down. Instead, select a tier and instance size that meet your performance requirements under typical load, and then scale out the instances to handle changes in traffic volume. Scaling up or down might trigger an application restart.

- Don't use the same storage account for logs and application data. Create a separate storage account for logs to help prevent logging from reducing application performance.

- Monitor performance. Use a performance monitoring service such as Application Insights or New Relic to monitor application performance and behavior under load. Performance monitoring gives you real-time insight into the application, and helps you diagnose issues and perform root-cause failure analysis.

For resiliency, availability, and reliability considerations, see the Reliability pillar.

Performance efficiency vs. security

If performance is so poor that the data is unusable, you can consider the data inaccessible. You need to do whatever you can do from a security perspective to make sure that your services have optimal uptime and performance.

A popular and effective method for enhancing availability and performance is *load balancing*. Load balancing is a method of distributing network traffic across servers that are part of a service. Load balancing improves performance because the processor, network, and memory overhead for serving requests are distributed across all the load-balanced servers. Employ load balancing whenever you can, as appropriate for your services. For information about load balancing scenarios, see Optimize uptime and performance.

Consider how the following security measures impact performance:

- To optimize performance and maximize availability, application code should first try to get OAuth access tokens silently from a cache before attempting to acquire a token from the identity provider. OAuth is a technological standard that allows you to securely share information between services without exposing your password.

- Make sure to integrate critical security alerts and logs into security information and event management (SIEM) systems without introducing a high volume of low-value data. Low-value data can increase SIEM costs and false positives and decrease performance. For more information, see Security logs and alerts using Azure services.

- Use Azure Active Directory Connect (Azure AD Connect) to synchronize your on-premises directory with your cloud directory. Various factors affect the performance of Azure AD Connect. Ensure Azure AD Connect has enough capacity to keep underperforming systems from impeding security and productivity.

Large or complex organizations that provision more than 100,000 objects should follow the recommendations to optimize their Azure AD Connect implementation. For more information, see What is hybrid identity with Azure Active Directory?

- To gain access to real-time performance information at the packet level, use packet capture to set alerts.

For other security considerations, see the Security pillar.

Mission-critical workload documentation

Learn about building highly reliable workloads on Microsoft Azure. The articles provide architectural guidance and a prescriptive approach for designing, building, and operating mission-critical workloads. For a given set of business requirements, an application should always be operational and available. While there are many approaches to achieving high reliability, one of the goals is to accelerate adoption toward cloud native solutions to derive maximum value from the Microsoft Cloud.

Get started

OVERVIEW

What is a mission-critical workload?

Design methodology

CONCEPT

Architecture pattern

Design principles

Cross-cutting concerns

Design areas

CONCEPT

Application design

Application platform

Data platform

Networking and connectivity

Health modeling

Deployment and testing

Security

Operational procedures

Reference examples

ARCHITECTURE

Baseline reference architecture

Baseline with network controls

Baseline in Azure landing zones

Reference implementations

DEPLOY

Mission-Critical Online

Mission-Critical Connected

Learn

TRAINING

Challenge Project - Design a mission-critical web application

Design a health model for your mission-critical workload

Continuously validate and test mission-critical workloads

Video library

VIDEO

What is a mission-critical workload?

Global distribution

Define a health model

Continuously validate your workload

Use multiple subscriptions

Development environments

Zero downtime deployments

Integration with Azure landing zones

VIDEO

Demo - Continuous validation with Azure Load Test and Azure Chaos Studio

Demo - Ephemeral dev environments and automated feature validation

Demo - Monitoring and health modeling

Assessment

HOW-TO GUIDE

Mission-critical assessment tool

Industry solutions

ARCHITECTURE

Carrier-grade for telecommunications

Architect perspectives

VIDEO

Azure Friday - Health modeling for mission-critical workloads on Azure

Azure Friday - Continuously validate and test your mission-critical Azure workloads

Azure Friday - Deploy your mission-critical workload in an Azure landing zone

Azure Enablement Show - Designing a mission-critical workload on Azure

Azure Enablement Show - Integrating a mission-critical workload with Azure landing zones

Mission-critical workloads

Article • 02/01/2023

This section strives to address the challenges of designing mission-critical workloads on Azure. The guidance is based on lessons learned from reviewing numerous customer applications and first-party solutions. This section provides actionable and authoritative guidance that applies Well-Architected best practices as the technical foundation for building and operating a highly reliable solution on Azure at-scale.

What is a mission-critical workload?

The term *workload* refers to a collection of application resources that support a common business goal or the execution of a common business process, with multiple services, such as APIs and data stores, working together to deliver specific end-to-end functionality.

The term *mission-critical* refers to a criticality scale that covers significant financial cost (business-critical) or human cost (safety-critical) associated with unavailability or underperformance.

A *mission-critical workload* therefore describes a collection of application resources, which must be highly reliable on the platform. The workload must always be available, resilient to failures, and operational.

Video: Mission-critical workloads on Azure

https://www.microsoft.com/en-us/videoplayer/embed/RE52ryK?postJsllMsg=true

What are the common challenges?

Microsoft Azure makes it easy to deploy and manage cloud solutions. However, building mission-critical workloads that are highly reliable on the platform remains a challenge for these main reasons:

- Designing a reliable application at scale is complex. It requires extensive platform knowledge to select the right technologies *and* optimally configure them to deliver end-to-end functionality.

- Failure is inevitable in any complex distributed system, and the solution must therefore be architected to handle failures with correlated or cascading impact.

This is a change in mindset for many developers and architects entering the cloud from an on-premises environment; reliability engineering is no longer an infrastructure subject, but should be a first-class concern within the application development process.

- Operationalizing mission-critical workloads requires a high degree of engineering rigor and maturity throughout the end-to-end engineering lifecycle as well as the ability to learn from failure.

Is mission-critical only about reliability?

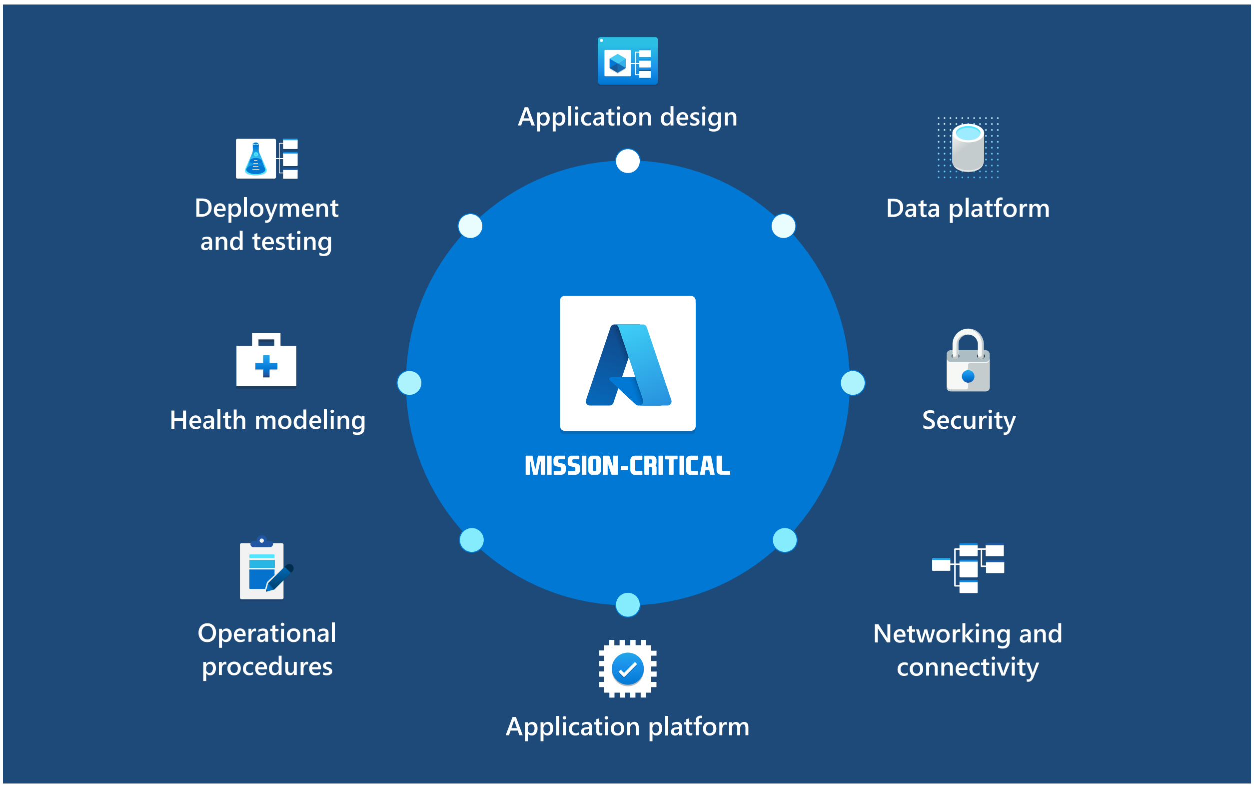
While the primary focus of mission-critical workloads is Reliability, other pillars of the Well-Architected Framework are equally important when building and operating a mission-critical workload on Azure.

- Security: how a workload mitigates security threats, such as Distributed Denial of Service (DDoS) attacks, will have a significant bearing on overall reliability.
- Operational Excellence: how a workload is able to effectively respond to operational issues will have a direct impact on application availability.
- Performance Efficiency: availability is more than simple uptime, but rather a consistent level of application service and performance relative to a known healthy state.

Achieving high reliability imposes significant cost tradeoffs, which may not be justifiable for every workload scenario. It is therefore recommended that design decisions be driven by business requirements.

What are the key design areas?

Mission-critical guidance within this series is composed of architectural considerations and recommendations orientated around these key design areas.



The design areas are interrelated and decisions made within one area can impact or influence decisions across the entire design. We recommend that readers familiarize themselves with these design areas, reviewing provided considerations and recommendations to better understand the consequences of encompassed decisions. For example, to define a target architecture it's critical to determine how best to monitor application health across key components. In this instance, the reader should review the **health modeling** design area, using the outlined recommendations to help drive decisions.

Design area	Summary
Application design	The use of a scale-unit architecture in the context of building a highly reliable application. Also explores the cloud application design patterns that allow for scaling, and error handling.
Application platform	Decision factors and recommendations related to the selection, design, and configuration of an appropriate application hosting platform, application dependencies, frameworks, and libraries.
Data platform	Choices in data store technologies, informed by evaluating the required—volume, velocity, variety, veracity.
Networking and connectivity	Network topology concepts at an application level, considering requisite connectivity and redundant traffic management. Critical recommendations intended to inform the design of a secure and scalable global network topology.

Design area	Summary
Health modeling and observability	Processes to define a robust health model, mapping quantified application health states through observability and operational constructs to achieve operational maturity.
Deployment and testing	Eradicate downtime and maintain application health for deployment operations, providing key considerations and recommendations intended to inform the design of optimal CI/CD pipelines for a mission-critical application.
Security	Protect the application against threats intended to directly or indirectly compromise its reliability.
Operational procedures	Adoption of DevOps and related deployment methods is used to drive effective and consistent operational procedures.

Illustrative examples

The guidance provided within this series is based on a solution-orientated approach to illustrate key design considerations and recommendations. There are several reference implementations available that can be used as a basis for further solution development.

- Baseline architecture of an internet-facing application—Provides a foundation for building a cloud-native, highly scalable, internet-facing application on Microsoft Azure. The workload is accessed over a public endpoint and doesn't require private network connectivity to a surrounding organizational technical estate.

  > Refer to the implementation: Mission-Critical Online

- Baseline architecture of an internet-facing application with network controls—Extends the baseline architecture with strict network controls in place to prevent unauthorized public access from the internet to any of the workload resources.

- Baseline architecture in an Azure landing zone—Provides a foundation for building a corporate-connected cloud-native application on Microsoft Azure using existing network infrastructure and private endpoints. The workload requires private connectivity to other organizational resources and takes a dependency on pre-provided Virtual Networks for connectivity to other organizational resources. This use case is intended for scenarios that require integration with a broader organizational technical estate for either public-facing or internal-facing workloads.

  > Refer to the implementation: Mission-Critical Connected

Industry scenarios

The mission-critical guidance within this series forms an industry agnostic design methodology which can be applied across a multitude of different industry contexts. The following list provides specific examples where the mission-critical design methodology has been applied and tailored to a particular industry scenario.

- Carrier-grade within the telecommunications industry

A carrier-grade workload pivots on both business-critical and safety-critical aspects, where there's a fundamental requirement to be operational with only minutes or even seconds of downtime per calendar year. Failure to achieve this uptime requirement can result in extensive loss of life, incur significant fines, or contractual penalties.

Next step

Start by reviewing the design methodology for mission-critical application scenarios.

Design methodology

Design methodology for mission-critical workloads on Azure

Article • 03/15/2023

Building a mission-critical application on any cloud platform requires significant technical expertise and engineering investment, particularly since there's significant complexity associated with:

- Understanding the cloud platform,
- Choosing the right services and composition,
- Applying the correct service configuration,
- Operationalizing utilized services, and
- Constantly aligning with the latest best practices and service roadmaps.

This design methodology strives to provide an easy to follow design path to help navigate this complexity and inform design decisions required to produce an optimal target architecture.

1—Design for business requirements

Not all mission-critical workloads have the same requirements. Expect that the review considerations and design recommendations provided by this design methodology will yield different design decisions and trade-offs for different application scenarios.

Select a reliability tier

Reliability is a relative concept and for any workload to be appropriately reliable it should reflect the business requirements surrounding it. For example, a mission-critical workload with a 99.999% availability Service Level Objective (SLO) requires a much higher level of reliability than another less critical workload with an SLO of 99.9%.

This design methodology applies the concept of reliability tiers expressed as availability SLOs to inform required reliability characteristics. The table below captures permitted error budgets associated with common reliability tiers.

Reliability Tier (Availability SLO)	Permitted Downtime (Week)	Permitted Downtime (Month)	Permitted Downtime (Year)
99.9%	10 minutes, 4 seconds	43 minutes, 49 seconds	8 hours, 45 minutes, 56 seconds

Reliability Tier (Availability SLO)	Permitted Downtime (Week)	Permitted Downtime (Month)	Permitted Downtime (Year)
99.95%	5 minutes, 2 seconds	21 minutes, 54 seconds	4 hours, 22 minutes, 58 seconds
99.99%	1 minutes	4 minutes 22 seconds	52 minutes, 35 seconds
99.999%	6 seconds	26 seconds	5 minutes, 15 seconds
99.9999%	<1 second	2 seconds	31 seconds

ⓘ **Important**

Availability SLO is considered by this design methodology to be more than simple uptime, but rather a consistent level of application service relative to a known healthy application state.

As an initial exercise, readers are advised to select a target reliability tier by determining how much downtime is acceptable? The pursuit of a particular reliability tier will ultimately have a significant bearing on the design path and encompassed design decisions, which will result in a different target architecture.

This image shows how the different reliability tiers and underlying business requirements influence the target architecture for a conceptual reference implementation, particularly concerning the number of regional deployments and utilized global technologies.

Recovery Time Objective (RTO) and Recovery Point Objective (RPO) are further critical aspects when determining required reliability. For instance, if you're striving to achieve an application RTO of less than a minute then back-up based recovery strategies or an active-passive deployment strategy are likely to be insufficient.

2—Evaluate the design areas using the design principles

At the core of this methodology lies a critical design path comprised of:

- Foundational **design principles**
- Fundamental **design area** with heavily interrelated and dependent design decisions.

The impact of decisions made within each design area will reverberate across other design areas and design decisions. Review the provided considerations and recommendations to better understand the consequences of encompassed decisions, which may produce trade-offs within related design areas.

For example, to define a target architecture it's critical to determine how best to monitor application health across key components. We highly recommend that you review the health modeling design area, using the outlined recommendations to help drive decisions.

3—Deploy your first mission-critical application

Refer to these reference architectures that describe the design decisions based on this methodology.

- Baseline architecture of an internet-facing application
- Baseline architecture of an internet-facing application with network controls

> **Tip**
>
> The architecture is backed by **Mission-Critical Online** implementation that illustrates the design recommendations.

Production-grade artifacts Every technical artifact is ready for use in production environments with all end-to-end operational aspects considered.

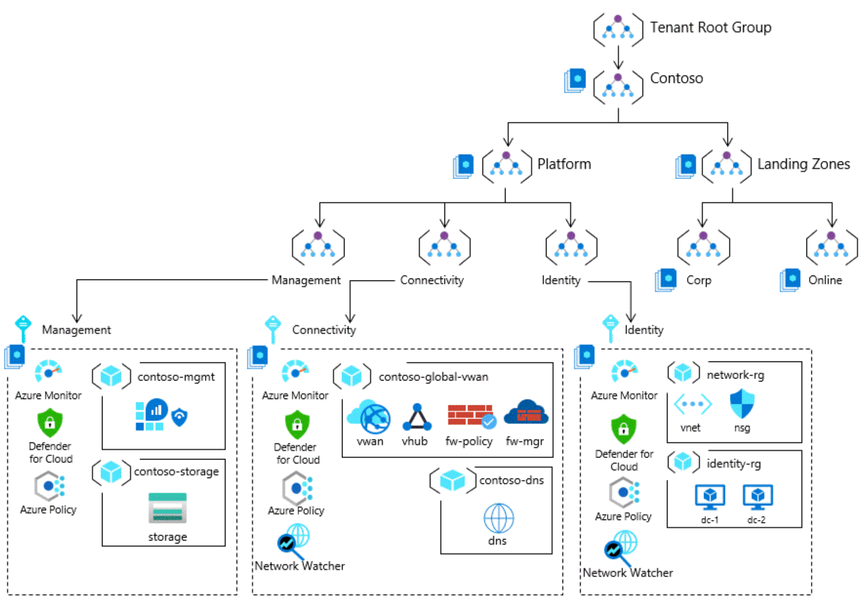
Rooted in real-world experiences All technical decisions are guided by experiences of Azure customers and lessons learned from deploying those solutions.

Azure roadmap alignment The mission-critical reference architectures have their own roadmap that is aligned with Azure product roadmaps.

4—Integrate your workload in Azure landing zones

Azure landing zone subscriptions provide shared infrastructure for enterprise deployments that need centralized governance. https://learn-video.azurefd.net/vod/player?id=9e05a6bd-7d10-4a83-9436-370a75dc1919&locale=en-us&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-design-methodology

It's crucial to evaluate which connectivity use case is required by your mission-critical application. Azure landing zones support two main archetypes separated into different Management Group scopes: **Online** or **Corp.** as shown in this image.



Online subscription

A mission-critical workload operates as an independent solution, without any direct corporate network connectivity to the rest of the Azure landing zone architecture. The application will be further safeguarded through the policy-driven governance and will automatically integrate with centralized platform logging through policy.

The baseline architecture and Mission-Critical Online implementation align with the Online approach.

Corp. subscription

When deployed in a Corp. subscription a mission-critical workload depends on the Azure landing zone to provide connectivity resources. This approach allows integration with other applications and shared services. You'll need to design around some foundational resources, which will exist up-front as part of the shared-service platform. For example, the regional deployment stamp should no longer encompass an ephemeral Virtual Network or Azure Private DNS Zone because these will exist in the Corp. subscription.

To get started with this use case, we recommend the **baseline architecture in an Azure landing zone** reference architecture.

> **Tip**
>
> The preceding architecture is backed by **Mission-Critical Connected** implementation.

5—Deploy a sandbox application environment

In parallel to design activities, it's highly recommended that a sandbox application environment is established using the Mission-Critical reference implementations.

This provides hands-on opportunities to validate design decisions by replicating the target architecture, allowing for design uncertainty to be quickly assessed. If applied correctly with representative requirement coverage, most problematic issues likely to hinder progress can be uncovered and subsequently addressed.

6—Continuously evolve with Azure roadmaps

Application architectures established using this design methodology must continue to evolve in alignment with Azure platform roadmaps to support optimized sustainability.

Next step

Review the design principles for mission-critical application scenarios.

Design principles

Design principles of a mission-critical workload

Article • 02/01/2023

The mission-critical design methodology is underpinned by five key design principles which serve as a compass for subsequent design decisions across the critical design areas. We highly recommend that you familiarize yourselves with these principles to better understand their impact and the trade-offs associated with non-adherence.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**

These mission-critical design principles resonate and extend the quality pillars of the Azure Well-Architected Framework—Reliability, Security, Cost Optimization, Operational Excellence, and Performance Efficiency.

Reliability

Maximum reliability - Fundamental pursuit of the most reliable solution, ensuring trade-offs are properly understood.

Design principle	Considerations
Active/Active design	To maximize availability and achieve regional fault tolerance, solution components should be distributed across multiple Availability Zones and Azure regions using an active/active deployment model where possible.
Blast radius reduction and fault isolation	Failure is impossible to avoid in a highly distributed multi-tenant cloud environment like Azure. By anticipating failures and correlated impact, from individual components to entire Azure regions, a solution can be designed and developed in a resilient manner.
Observe application health	Before issues impacting application reliability can be mitigated, they must first be detected and understood. By monitoring the operation of an application relative to a known healthy state it becomes possible to detect or even predict reliability issues, allowing for swift remedial action to be taken.
Drive automation	One of the leading causes of application downtime is human error, whether that is due to the deployment of insufficiently tested software or misconfiguration. To minimize the possibility and impact of human errors, it's vital to strive for automation in all aspects of a cloud solution to improve reliability; automated testing, deployment, and management.
Design for self-healing	Self healing describes a system's ability to deal with failures automatically through pre-defined remediation protocols connected to failure modes within the solution. It's an advanced concept that requires a high level of system maturity with monitoring and automation, but should be an aspiration from inception to maximize reliability.
Complexity avoidance	Avoid unnecessary complexity when designing the solution and all operational processes to drive reliability and management efficiencies, minimizing the likelihood of failures.

Performance Efficiency

Sustainable performance and scalability - Design for scalability across the end-to-end solution without performance bottlenecks.

Design principle	Considerations
Design for scale-out	Scale-out is a concept that focuses on a system's ability to respond to demand through horizontal growth. This means that as traffic grows, more resource units are added in parallel instead of increasing the size of the existing resources. A systems ability to handle expected and unexpected traffic increases through scale-units is essential to overall performance and reliability by further reducing the impact of a single resource failure.

Design principle	Considerations
Automation for hyperscale	Scale operations throughout the solution should be fully automated to minimize the performance and availability impact from unexpected or expected increases in traffic, ensuring the time it takes to conduct scale operations is understood and aligned with a model for application health.
Continuous validation and testing	Automated testing should be performed within CI/CD processes to drive continuous validation for each application change. Load testing against a performance baseline with synchronized chaos experimentation should be included to validate existing thresholds, targets, and assumptions, as well as helping to quickly identify risks to resiliency and availability. Such testing should be conducted within staging and testing environments, but also optionally within development environments. It can also be beneficial to run a subset of tests against the production environment, particularly in conjunction with a blue/green deployment model to validate new deployment stamps before receiving production traffic.
Reduce overhead with managed compute services	Using managed compute services and containerized architectures significantly reduces the ongoing administrative and operational overhead of designing, operating, and scaling applications by shifting infrastructure deployment and maintenance to the managed service provider.
Baseline performance and identify bottlenecks	Performance testing with detailed telemetry from every system component allows for the identification of bottlenecks within the system, including components that need to be scaled in relation to other components, and this information should be incorporated into a capacity model.
Model capacity	A capacity model enables planning of resource scale levels for a given load profile, and additionally exposes how system components perform in relation to each other, therefore enabling system-wide capacity allocation planning.

Operational Excellence

Operations by design - Engineered to last with robust and assertive operational management.

Design principle	Considerations
Loosely coupled components	Loose coupling enables independent and on-demand testing, deployments, and updates to components of the application while minimizing inter-team dependencies for support, services, resources, or approvals.

Design principle	Considerations
Automate build and release processes	Fully automated build and release processes reduce the friction and increase the velocity of deploying updates, bringing repeatability and consistency across environments. Automation shortens the feedback loop from developers pushing changes to getting insights on code quality, test coverage, resiliency, security, and performance, which increases developer productivity.
Developer agility	Continuous Integration and Continuous Deployment (CI/CD) automation enables the use of short-lived development environments with lifecycles tied to that of an associated feature branch, which promotes developer agility and drives validation as early as possible within the engineering cycle to minimize the engineering cost of bugs.
Quantify operational health	Full diagnostic instrumentation of all components and resources enables ongoing observability of logs, metrics and traces, but also facilitates health modeling to quantify application health in the context to availability and performance requirements.
Rehearse recovery and practice failure	Business Continuity (BC) and Disaster Recovery (DR) planning and practice drills are essential and should be conducted frequently, since learnings can iteratively improve plans and procedures to maximize resiliency in the event of unplanned downtime.
Embrace continuous operational improvement	Prioritize routine improvement of the system and user experience, using a health model to understand and measure operational efficiency with feedback mechanisms to enable application teams to understand and address gaps in an iterative manner.

Security

Always secure - Design for end-to-end security to maintain application stability and ensure availability.

Design principle	Considerations
Monitor the security of the entire solution and plan incident responses	Correlate security and audit events to model application health and identify active threats. Establish automated and manual procedures to respond to incidents using Security Information and Event Management (SIEM) tooling for tracking.

Design principle	Considerations
Model and test against potential threats	Ensure appropriate resource hardening and establish procedures to identify and mitigate known threats, using penetration testing to verify threat mitigation, as well as static code analysis and code scanning.
Identify and protect endpoints	Monitor and protect the network integrity of internal and external endpoints through security capabilities and appliances, such as firewalls or web application firewalls. Use industry standard approaches to protect against common attack vectors like Distributed Denial-Of-Service (DDoS) attacks, such as SlowLoris.
Protect against code level vulnerabilities	Identify and mitigate code-level vulnerabilities, such as cross-site scripting or SQL injection, and incorporate security patching into operational lifecycles for all parts of the codebase, including dependencies.
Automate and use least privilege	Drive automation to minimize the need for human interaction and implement least privilege across both the application and control plane to protect against data exfiltration and malicious actor scenarios.
Classify and encrypt data	Classify data according to risk and apply industry standard encryption at rest and in transit, ensuring keys and certificates are stored securely and managed properly.

Cost Optimization

There are obvious cost tradeoffs associated with introducing greater reliability, which should be carefully considered in the context of workload requirements.

Maximizing reliability can impact the overall financial cost of the solution. For example, the duplication of resources and the distribution of resources across regions to achieve high availability has clear cost implications. To avoid excess costs, don't over-engineer or over-provision beyond the relevant business requirements.

Also, there is added cost associated with engineering investment in fundamental reliability concepts, such as embracing infrastructure as code, deployment automation, and chaos engineering. This comes at a cost in terms of both time and effort, which could be invested elsewhere to deliver new application functionality and features.

Cloud native design

- Azure-native managed services - Azure-native managed services are prioritized due to their lower administrative and operational overhead as well as tight integration with consistent configuration and instrumentation across the application stack.

- **Roadmap alignment** - Incorporate upcoming new and improved Azure service capabilities as they become Generally Available (GA) to stay close to the leading edge of Azure.

- **Embrace preview capabilities and mitigate known gaps** - While Generally Available (GA) services are prioritized for supportability, Azure service previews are actively explored for rapid incorporation, providing technical and actionable feedback to Azure product groups to address gaps.

- **Azure landing zone alignment** - Deployable within an Azure landing zone and aligned to the Azure landing zone design methodology, but also fully functional and deployable in a bare environment outside of a landing zone.

Next step

Review cross-cutting concerns associated with mission-critical workloads.

Cross-cutting concerns

Architecture pattern for mission-critical workloads on Azure

Article • 12/16/2022

This article presents a key pattern for mission-critical architectures on Azure. Apply this pattern when you start your design process, and then select components that are best suited for your business requirements. The article recommends a *north star* design approach and includes other examples with common technology components.

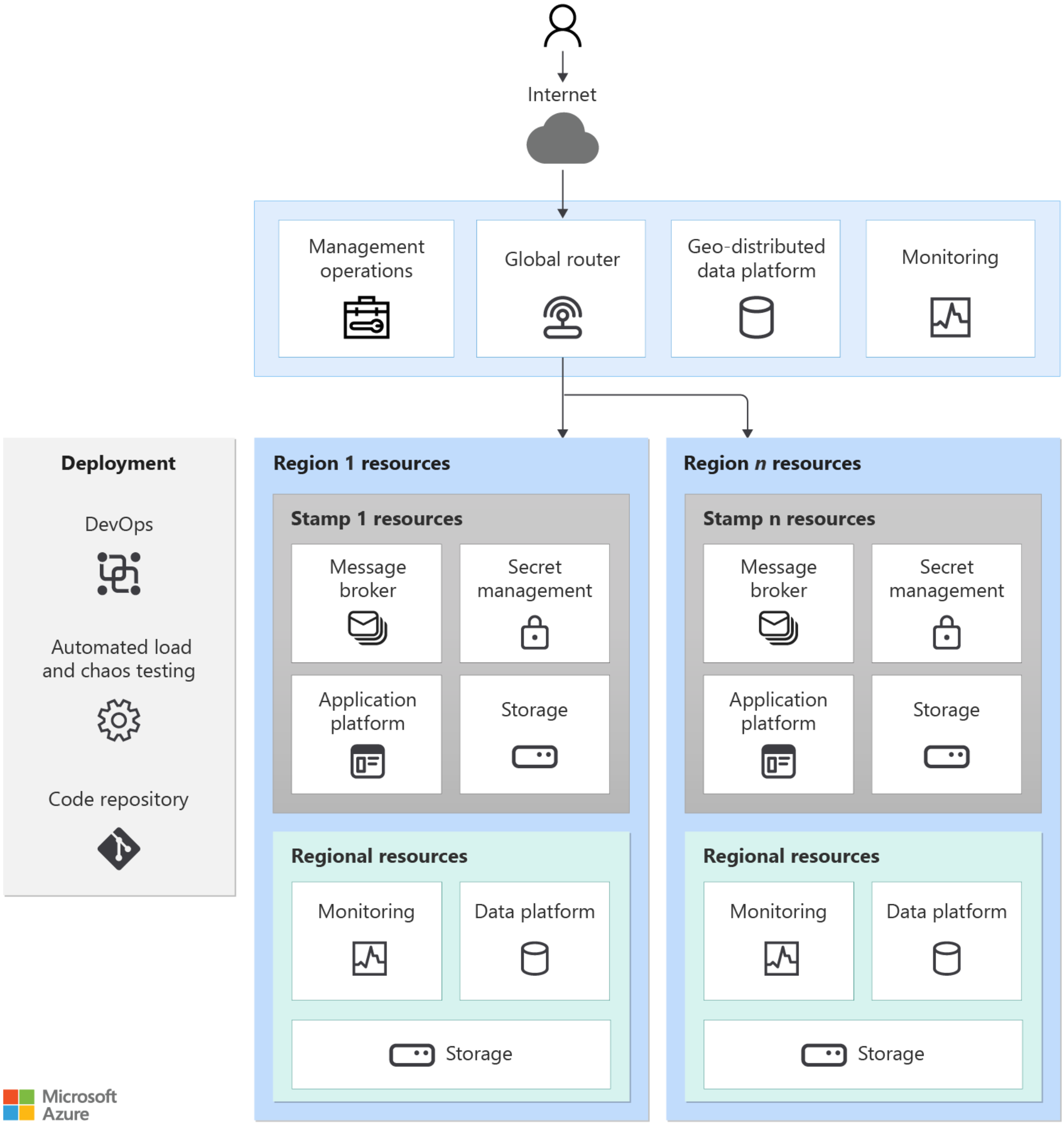
We recommend that you evaluate the key design areas, define the critical user and system flows that use the underlying components, and develop a matrix of Azure resources and their configuration while keeping in mind the following characteristics.

Characteristic	Considerations
Lifetime	What's the expected lifetime of the resource, relative to other resources in the solution? Should the resource outlive or share the lifetime with the entire system or region, or should it be temporary?
State	What impact will the persisted state at this layer have on reliability or manageability?
Reach	Is the resource required to be globally distributed? Can the resource communicate with other resources, located globally or within that region?
Dependencies	What are the dependencies on other resources?
Scale limits	What is the expected throughput for that resource? How much scale is provided by the resource to fit that demand?
Availability/disaster recovery	What is the impact on availability from a disaster at this layer? Would it cause a systemic outage or only a localized capacity or availability issue?

ⓘ **Important**

This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**

Core architecture pattern



Global resources

Certain resources are globally shared by resources deployed within each region. Common examples are resources that are used to distribute traffic across multiple regions, store permanent state for the whole application, and monitor resources for them.

Characteristic	Considerations
Lifetime	These resources are expected to be long living (non-ephemeral). Their lifetime spans the life of the system or longer. Often the resources are managed with in-place data and control plane updates, assuming they support zero-downtime update operations.

Characteristic	Considerations
State	Because these resources exist for at least the lifetime of the system, this layer is often responsible for storing global, geo-replicated state.
Reach	The resources should be globally distributed and replicated to the regions that host those resources. It’s recommended that these resources communicate with regional or other resources with low latency and the desired consistency.
Dependencies	The resources should avoid dependencies on regional resources because their unavailability can be a cause for global failure. For example, certificates or secrets kept in a single vault could have global impact if there's a regional failure where the vault is located.
Scale limits	Often these resources are singleton instances in the system, and they should be able to scale such that they can handle throughput of the system as a whole.
Availability/disaster recovery	Regional and stamp resources can use global resources. It's critical that global resources are configured with high availability and disaster recovery for the health of the whole system.

Regional stamp resources

The stamp contains the application and resources that participate in completing business transactions. A stamp typically corresponds to a deployment to an Azure region. Although a region can have more than one stamp.

Characteristic	Considerations
Lifetime	The resources are expected to have a short life span (ephemeral) with the intent that they can get added and removed dynamically while regional resources outside the stamp continue to persist. The ephemeral nature is needed to provide more resiliency, scale, and proximity to users.
State	Because stamps are ephemeral and will be destroyed with each deployment, a stamp should be stateless as much as possible.
Reach	Can communicate with regional and global resources. However, communication with other regions or other stamps should be avoided.
Dependencies	The stamp resources must be independent. They're expected to have regional and global dependencies but shouldn't rely on components in other stamps in the same or other regions.

Characteristic	Considerations
Scale limits	Throughput is established through testing. The throughput of the overall stamp is limited to the least performant resource. Stamp throughput needs to estimate the high-level of demand caused by a failover to another stamp.
Availability/disaster recovery	Because of the temporary nature of stamps, disaster recovery is done by redeploying the stamp. If resources are in an unhealthy state, the stamp, as a whole, can be destroyed and redeployed.

Regional resources

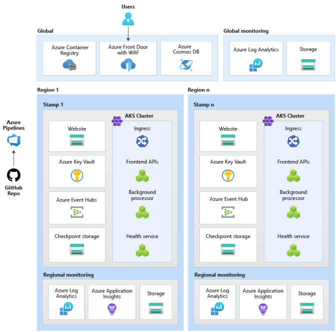
A system can have resources that are deployed in region but outlive the stamp resources. For example, observability resources that monitor resources at the regional level, including the stamps.

Characteristic	Consideration
Lifetime	The resources share the lifetime of the region and out live the stamp resources.
State	State stored in a region can't live beyond the lifetime of the region. If state needs to be shared across regions, consider using a global data store.
Reach	The resources don't need to be globally distributed. Direct communication with other regions should be avoided at all cost.
Dependencies	The resources can have dependencies on global resources, but not on stamp resources because stamps are meant to be short lived.
Scale limits	Determine the scale limit of regional resources by combining all stamps within the region.

Baseline architectures for mission-critical workloads

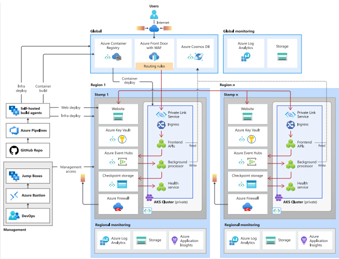
These baseline examples serve as the recommended north star architecture for mission-critical applications. The baseline strongly recommends containerization and using a container orchestrator for the application platform. The baseline uses Azure Kubernetes Service (AKS).

> Refer to Well-Architected mission-critical workloads: Containerization.



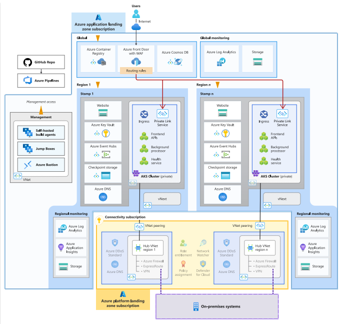
Baseline architecture

If you're just starting your mission-critical journey, use this architecture as a reference. The workload is accessed over a public endpoint and doesn't require private network connectivity to other company resources.



Baseline with network controls

This architecture builds on the baseline architecture. The design is extended to provide strict network controls to prevent unauthorized public access from the internet to the workload resources.



Baseline in Azure landing zones

This architecture is appropriate if you're deploying the workload in an enterprise setup where integration within a broader organization is required. The workload uses centralized shared services, needs on-premises connectivity, and integrates with other workloads within the enterprise. It's deployed in an Azure landing zone subscription that inherits from the Corp. management group.

Design areas

We recommend that you use the provided design guidance to navigate the key design decisions to reach an optimal solution. For information, see What are the key design areas?

Next step

Review the best practices for designing mission-critical application scenarios.

Application design

Cross-cutting concerns of mission-critical workloads on Azure

Article • 02/01/2023

There are several cross-cutting concerns that traverse the key design areas. This article contextualizes these cross-cutting concerns for subsequent consideration within each design area.

ⓘ **Important**

This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**

Scale limits

Azure applies various *limits* or *quotas* to ensure a consistent level of service for all customers. Examples of these limits include restrictions on the number of deployable resources within a single subscription, and restrictions to network and query throughput.

Service limits may have a significant bearing on a large mission-critical workload. Consider the limits of the services used in the target architecture carefully to ensure sustainable scale. Otherwise, you may hit one or more of these limits as the workload grows.

ⓘ **Important**

Limits and quotas may change as the platform evolves. Be sure to check the current limits at **Azure subscription and service limits, quotas, and constraints**.

Recommendations

- Employ a scale unit approach for resource composition, deployment, and management.
- Use subscriptions as scale units, scaling out resources and subscriptions as required.
- Ensure scale limits are considered as part of capacity planning.

- If available, use data about existing application environments to explore which limits might be encountered.

Automation

A holistic approach to automation of deployment and management activities can maximize the reliability and operability of the workload.

Recommendations

- Automate continuous integration and continuous delivery (CI/CD) pipelines for all application components.
- Automate application management activities, such as patching and monitoring.
- Use declarative management semantics, such as Infrastructure as code (IaC), instead of over imperative approaches.
- Prioritize templating over scripting. Defer to scripting only when using templates isn't possible.

Azure roadmap alignment

Azure is constantly evolving through frequent updates to services, features, and regional availability. It's important to align the target architecture with Azure platform roadmaps to inform an optimal application trajectory. For example, making sure that the required services and features are available within the chosen deployment regions.

Refer to Azure updates for the latest information about new services and features.

Recommendations

- Align with Azure engineering roadmaps and regional rollout plans.
- Unblock with preview services or by taking dependencies on the Azure platform roadmap.
- Only take a dependency on committed services and features; validate roadmap dependencies with Microsoft engineering product groups.

Next step

Explore the design areas that provide critical considerations and recommendations for building a mission-critical workload.

Architecture pattern

Application design of mission-critical workloads on Azure

Article • 04/10/2023

When you design an application, both functional and non-functional application requirements are critical. This design area describes architecture patterns and scaling strategies that can help make your application resilient to failures.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected Framework mission-critical workload** series. If you aren't familiar with this series, we recommend that you start with **What is a mission-critical workload?**.

Scale-unit architecture

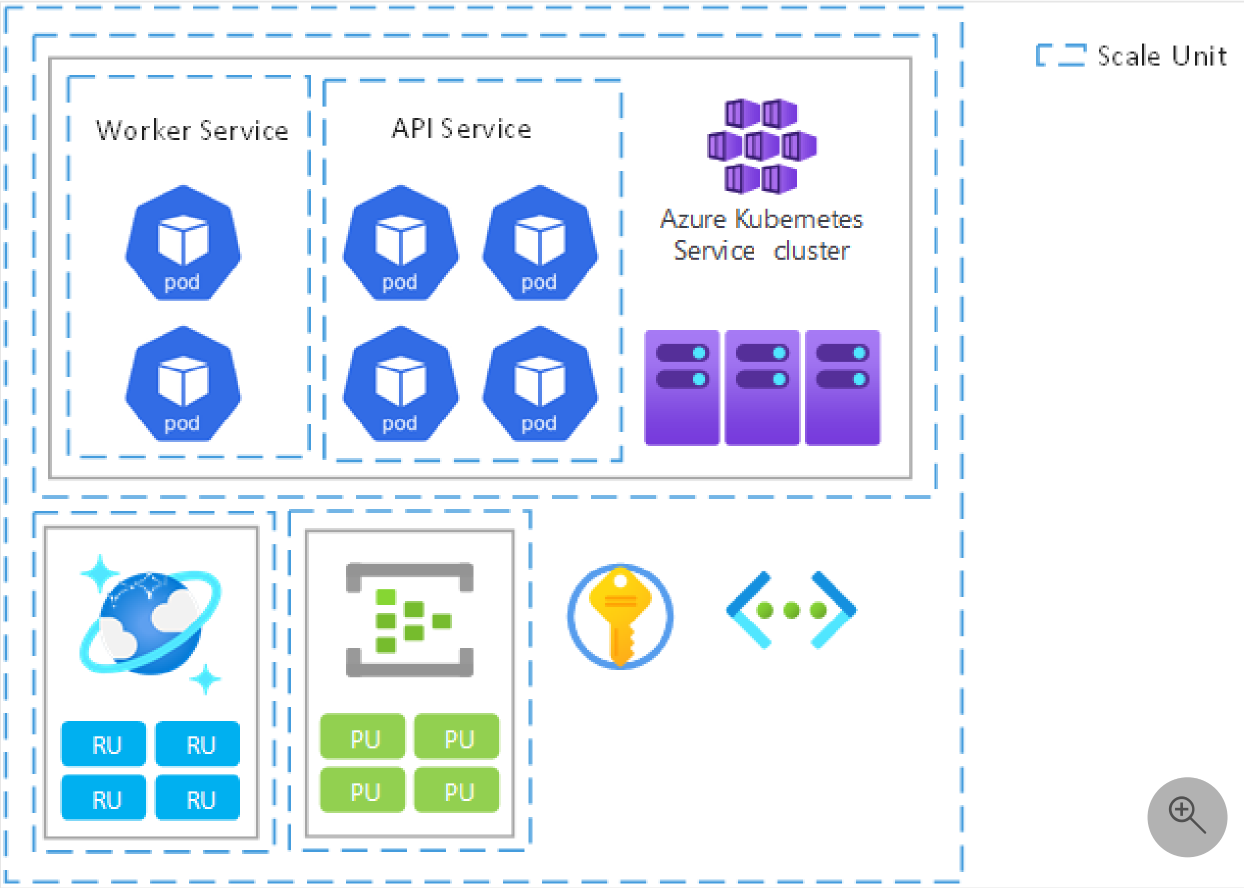
All functional aspects of a solution must be capable of scaling to meet changes in demand. We recommend that you use a scale-unit architecture to optimize end-to-end scalability through compartmentalization, and also to standardize the process of adding and removing capacity. A *scale unit* is a logical unit or function that can be scaled independently. A unit can be made up of code components, application hosting platforms, the deployment stamps that cover the related components, and even subscriptions to support multi-tenant requirements.

We recommend this approach because it addresses the scale limits of individual resources and the entire application. It helps with complex deployment and update scenarios because a scale unit can be deployed as one unit. Also, you can test and validate specific versions of components in a unit before directing user traffic to it.

Suppose your mission-critical application is an online product catalog. It has a user flow for processing product comments and ratings. The flow uses APIs to retrieve and post comments and ratings, and supporting components like an OAuth endpoint, datastore, and message queues. The stateless API endpoints represent granular functional units that must adapt to changes on demand. The underlying application platform must also be able to scale accordingly. To avoid performance bottlenecks, the downstream components and dependencies must also scale to an appropriate degree. They can either scale independently, as separate scale units, or together, as part of a single logical unit.

Example scale units

The following image shows the possible scopes for scale units. The scopes range from microservice pods to cluster nodes and regional deployment stamps.



Design considerations

- **Scope**. The scope of a scale unit, the relationship between scale units, and their components should be defined according to a capacity model. Take into consideration non-functional requirements for performance.

- **Scale limits**. Azure subscription scale limits and quotas might have a bearing on application design, technology choices, and the definition of scale units. Scale units can help you bypass the scale limits of a service. For example, if an AKS cluster in one unit can have only 1,000 nodes, you can use two units to increase that limit to 2,000 nodes.

- **Expected load**. Use the number of requests for each user flow, the expected peak request rate (requests per second), and daily/weekly/seasonal traffic patterns to inform core scale requirements. Also factor in the expected growth patterns for both traffic and data volume.

- **Acceptable degraded performance**. Determine whether a degraded service with high response times is acceptable under load. When you're modeling required capacity, the required performance of the solution under load is a critical factor.
- **Non-functional requirements**. Technical and business scenarios have distinct considerations for resilience, availability, latency, capacity, and observability. Analyze these requirements in the context of key end-to-end user flows. You'll have relative flexibility in the design, decision making, and technology choices at a user-flow level.

Design recommendations

- Define the scope of a scale unit and the limits that will trigger the unit to scale.
- Ensure that all application components can scale independently or as part of a scale unit that includes other related components.
- Define the relationship between scale units, based on a capacity model and non-functional requirements.
- Define a regional deployment stamp to unify the provisioning, management, and operation of regional application resources into a heterogenous but interdependent scale unit. As load increases, extra stamps can be deployed, within the same Azure region or different ones, to horizontally scale the solution.
- Use an Azure subscription as the scale unit so that scale limits within a single subscription don't constrain scalability. This approach applies to high-scale application scenarios that have significant traffic volume.
- Model required capacity around identified traffic patterns to make sure sufficient capacity is provisioned at peak times to prevent service degradation. Alternatively, optimize capacity during off-peak hours.
- Measure the time required to do scale-out and scale-in operations to ensure that the natural variations in traffic don't create an unacceptable level of service degradation. Track the scale operation durations as an operational metric.

> **Note**
>
> When you deploy in an Azure landing zone, ensure that the landing zone subscription is dedicated to the application to provide a clear management boundary and to avoid the **Noisy Neighbor antipattern**.

Global distribution

It's impossible to avoid failure in any highly distributed environment. This section provides strategies to mitigate many fault scenarios. The application must be able to withstand regional and zonal failures. It must be deployed in an active/active model so that the load is distributed among all regions.

Watch this video to get an overview of how to plan for failures in mission-critical applications and maximize resiliency:

https://learn-video.azurefd.net/vod/player?id=7cea20d8-8265-4c5c-aaba-5e174731c2e3&locale=en-us&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-application-design

Design considerations

- **Redundancy**. Your application must be deployed to multiple regions. Additionally, within a region, we strongly recommend that you use availability zones to allow for fault tolerance at the datacenter level. Availability zones have a latency perimeter of less than 2 milliseconds between availability zones. For workloads that are "chatty" across zones, this latency can introduce a performance penalty and incur bandwidth charges for interzone data transfer.

- **Active/active model**. An active/active deployment strategy is recommended because it maximizes availability and provides a higher composite service-level agreement (SLA). However, it can introduce challenges around data synchronization and consistency for many application scenarios. Address the challenges at a data platform level while considering the trade-offs of increased cost and engineering effort.

  An active/active deployment across multiple cloud providers is a way to potentially mitigate dependency on global resources within a single cloud provider. However, a multicloud active/active deployment strategy introduces a significant amount of complexity around CI/CD. Also, given the differences in resource specifications and capabilities among cloud providers, you'd need specialized deployment stamps for each cloud.

- **Geographical distribution**. The workload might have compliance requirements for geographical data residency, data protection, and data retention. Consider whether there are specific regions where data must reside or where resources need to be deployed.

- **Request origin.** The geographic proximity and density of users or dependent systems should inform design decisions about global distribution.

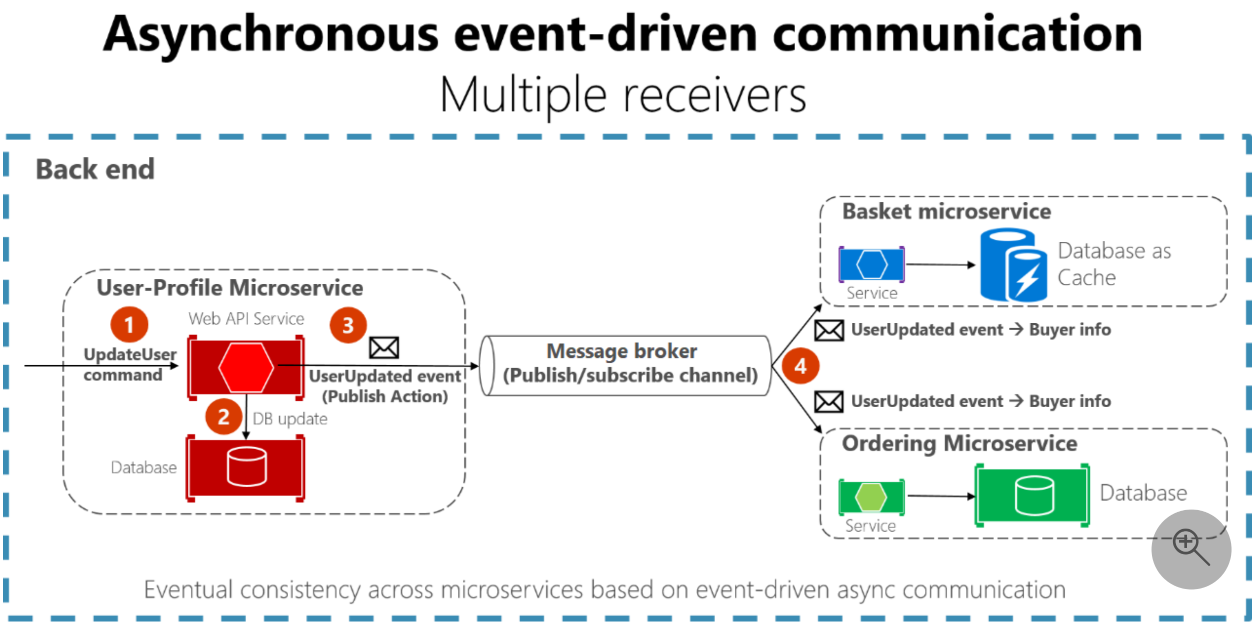
- **Connectivity**. How the workload is accessed by users or external systems will influence your design. Consider whether the application is available over the public internet or private networks that use either VPN or Azure ExpressRoute circuits.

For design recommendations and configuration choices at the platform level, see Application platform: Global distribution.

Loosely coupled event-driven architecture

Coupling enables interservice communication via well-defined interfaces. A *loose* coupling allows an application component to operate independently. A microservices architecture style is consistent with mission-critical requirements. It facilitates high availability by preventing cascading failures.

For loose coupling, we strongly recommend that you incorporate event-driven design. Asynchronous message processing through an intermediary can build resiliency.



In some scenarios, applications can combine loose and tight coupling, depending on business objectives.

Design considerations

- **Runtime dependencies**. Loosely coupled services shouldn't be constrained to use the same compute platform, programming language, runtime, or operating system.

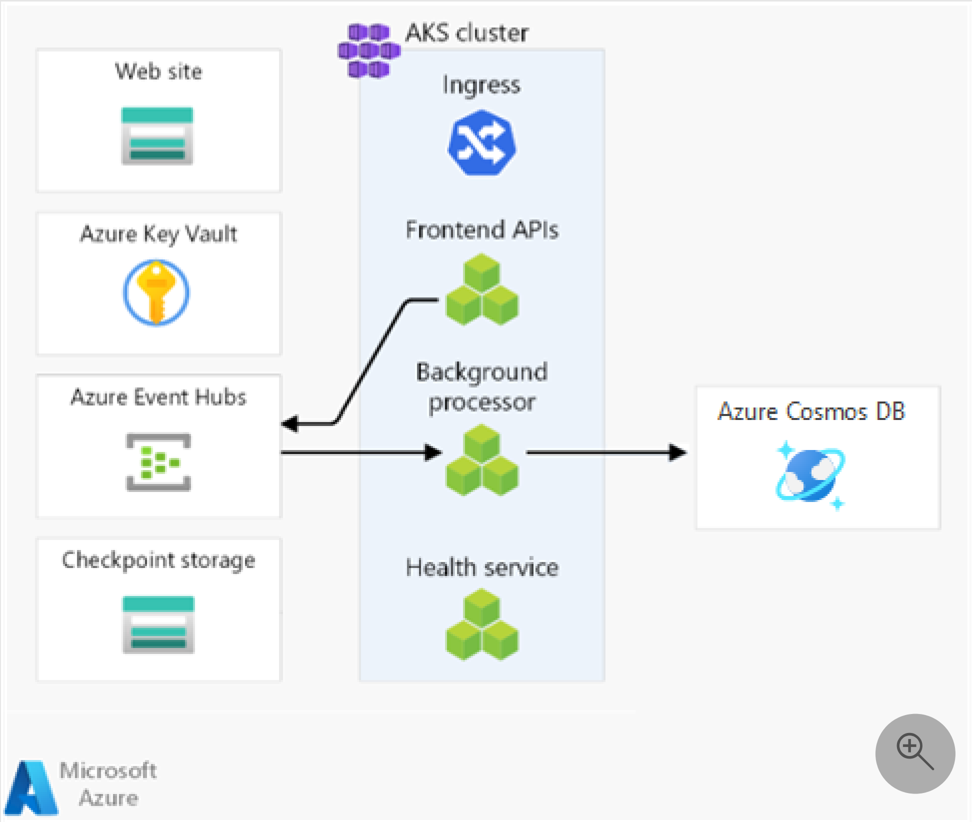
- **Scaling**. Services should be able to scale independently. Optimize the use of infrastructure and platform resources.
- **Fault tolerance**. Failures should be handled separately and shouldn't affect client transactions.
- **Transactional integrity**. Consider the effect of data creation and persistence that happens in separate services.
- **Distributed tracing**. End-to-end tracing might require complex orchestration.

Design recommendations

- Align microservice boundaries with critical user flows.
- Use event-driven asynchronous communication where possible to support sustainable scale and optimal performance.
- Use patterns like Outbox and Transactional Session to guarantee consistency so that every message is processed correctly.

Example: Event-driven approach

The Mission-Critical Online reference implementation uses microservices to process a single business transaction. It applies write operations asynchronously with a message broker and worker. Read operations are synchronous, with the result directly returned to the caller.



Resiliency patterns and error handling in application code

A mission-critical application must be designed to be resilient so that it addresses as many failure scenarios as possible. This resiliency maximizes service availability and reliability. The application should have self-healing capabilities, which you can implement by using design patterns like Retries with Backoff and Circuit Breaker.

For non-transient failures that you can't fully mitigate in application logic, the health model and operational wrappers need to take corrective action. Application code must incorporate proper instrumentation and logging to inform the health model and facilitate subsequent troubleshooting or root cause analysis as required. You need to implement distributed tracing to provide the caller with a comprehensive error message that includes a correlation ID when a failure occurs.

Tools like Application Insights can help you query, correlate, and visualize application traces.

Design considerations

- **Proper configurations**. It's not uncommon for transient problems to cause cascading failures. For example, retry without appropriate back-off exacerbates the problem when a service is being throttled. You can space retry delays linearly or increase them exponentially to back off through growing delays.

- **Health endpoints**. You can expose functional checks within application code by using health endpoints that external solutions can poll to retrieve application component health status.

Design recommendations

Here are some common software engineering patterns for resilient applications:

Pattern	Summary
Queue-Based Load Leveling	Introduces a buffer between consumers and requested resources to ensure consistent load levels. As consumer requests are queued, a worker process handles them against the requested resource at a pace that's set by the worker and by the requested resource's ability to process the requests. If consumers expect replies to their requests, you need to implement a separate response mechanism. Apply a prioritized order so that the most important activities are performed first.
Circuit Breaker	Provides stability by either waiting for recovery or quickly rejecting requests rather than blocking while waiting for an unavailable remote service or resource. This pattern also handles faults that might take a variable amount of time to recover from when a connection is made to a remote service or resource.
Bulkhead	Attempts to partition service instances into groups based on load and availability requirements, isolating failures to sustain service functionality.
Saga	Manages data consistency across microservices that have independent datastores by ensuring that services update each other through defined event or message channels. Each service performs local transactions to update its own state and publishes an event to trigger the next local transaction in the saga. If a service update fails, the saga runs compensating transactions to counteract preceding service update steps. Individual service update steps can themselves implement resiliency patterns, such as retry.
Health Endpoint Monitoring	Implements functional checks in an application that external tools can access through exposed endpoints at regular intervals. You can interpret responses from the endpoints by using key operational metrics to inform application health and trigger operational responses, like raising an alert or performing a compensating rollback deployment.

Pattern	Summary
Retry	Handles transient failures elegantly and transparently. - Cancel if the fault is unlikely to be transient and is unlikely to succeed if the operation is attempted again. - Retry if the fault is unusual or rare and the operation is likely to succeed if attempted again immediately. - Retry after a delay if the fault is caused by a condition that might need a short time to recover, like network connectivity or high-load failures. Apply a suitable back-off strategy as retry delays increase.
Throttling	Controls the consumption of resources used by application components, protecting them from becoming over-encumbered. When a resource reaches a load threshold, it defers lower-priority operations and degrading non-essential functionality so that essential functionality can continue until sufficient resources are available to return to normal operation.

Here are some additional recommendations:

- Use vendor-provided SDKs, like the Azure SDKs, to connect to dependent services. Use built-in resiliency capabilities instead of implementing custom functionality.
- Apply a suitable back-off strategy when retrying failed dependency calls to avoid a self-inflicted DDoS scenario.
- Define common engineering criteria for all application microservice teams to drive consistency and speed in the use of application-level resiliency patterns.
- Implement resiliency patterns by using proven standardized packages, like Polly for C# or Sentinel for Java.
- Use correlation IDs for all trace events and log messages to link them to a given request. Return correlation IDs to the caller for all calls, not just failed requests.
- Use structured logging for all log messages. Select a unified operational data sink for application traces, metrics, and logs to enable operators to easily debug problems. For more information, see Collect, aggregate, and store monitoring data for cloud applications.
- Ensure that operational data is used together with business requirements to inform an application health model.

Programming language selection

It's important to select the right programming languages and frameworks. These decisions are often driven by the skill sets or standardized technologies in the organization. However, it's essential to evaluate the performance, resilience, and overall capabilities of various languages and frameworks.

Design considerations

- **Development kit capabilities**. There are differences in the capabilities that are offered by Azure service SDKs in various languages. These differences might influence your choice of an Azure service or programming language. For example, if Azure Cosmos DB is a feasible option, Go might not be an appropriate development language because there's no first-party SDK.

- **Feature updates**. Consider how often the SDK is updated with new features for the selected language. Commonly used SDKs, like .NET and Java libraries, are updated frequently. Other SDKs or SDKs for other languages might be updated less frequently.

- **Multiple programming languages or frameworks**. You can use multiple technologies to support various composite workloads. However, you should avoid sprawl because it introduces management complexity and operational challenges.

- **Compute option**. Legacy or proprietary software might not run in PaaS services. Also, you might not be able to include legacy or proprietary software in containers.

Design recommendations

- Evaluate all relevant Azure SDKs for the capabilities you need and your chosen programming languages. Verify alignment with non-functional requirements.

- Optimize the selection of programming languages and frameworks at the microservice level. Use multiple technologies as appropriate.

- Prioritize the .NET SDK to optimize reliability and performance. .NET Azure SDKs typically provide more capabilities and are updated frequently.

Next step

Review the considerations for the application platform.

Application platform

Application platform considerations for mission-critical workloads on Azure

Article • 02/01/2023

Azure provides many compute services for hosting highly available applications. The services differ in capability and complexity. We recommend that you choose services based on:

- Non-functional requirements for reliability, availability, performance, and security.
- Decision factors like scalability, cost, operability, and complexity.

The choice of an application hosting platform is a critical decision that affects all other design areas. For example, legacy or proprietary development software might not run in PaaS services or containerized applications. This limitation would influence your choice of compute platform.

A mission-critical application can use more than one compute service to support multiple composite workloads and microservices, each with distinct requirements.

This design area provides recommendations related to compute selection, design, and configuration options. We also recommend that you familiarize yourself with the Compute decision tree.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected Framework mission-critical workload** series. If you aren't familiar with this series, we recommend that you start with **What is a mission-critical workload?**.

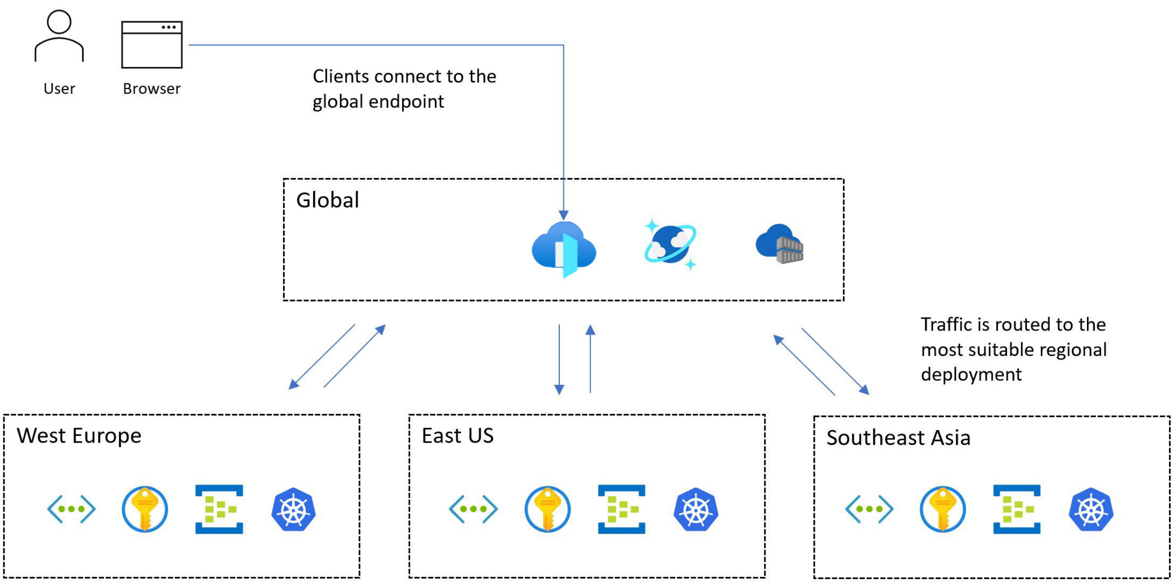
Global distribution of platform resources

A typical pattern for a mission-critical workload includes global resources and regional resources.

Azure services, which aren't constrained to a particular Azure region, are deployed or configured as global resources. Some use cases include distributing traffic across multiple regions, storing permanent state for a whole application, and caching global static data. If you need to accommodate both a scale-unit architecture and global distribution, consider how resources are optimally distributed or replicated across Azure regions.

Other resources are deployed regionally. These resources, which are deployed as part of a deployment stamp, typically correspond to a scale unit. However, a region can have more than one stamp, and a stamp can have more than one unit. The reliability of regional resources is crucial because they're responsible for running the main workload.

The following image shows the high-level design. A user accesses the application via a central global entry point that then redirects requests to a suitable regional deployment stamp:



The mission-critical design methodology requires a multi-region deployment. This model ensures regional fault tolerance, so that the application remains available even when an entire region goes down. When you design a multi-region application, consider different deployment strategies, like active/active and active/passive, together with application requirements, because there are significant trade-offs for each approach. For mission-critical workloads, we strongly recommend the active/active model.

Not every workload supports or requires running multiple regions simultaneously. You should weigh specific application requirements against trade-offs to determine an optimal design decision. For certain application scenarios that have lower reliability targets, active/passive or sharding can be suitable alternatives.

Availability zones can provide highly available regional deployments across different datacenters within a region. Nearly all Azure services are available in either a zonal configuration, where the service is delegated to a specific zone, or a zone-redundant configuration, where the platform automatically ensures that the service spans across zones and can withstand a zone outage. These configurations provide fault tolerance up to the datacenter level.

Design considerations

- **Regional and zonal capabilities**. Not all services and capabilities are available in every Azure region. This consideration could affect the regions you choose. Also, availability zones aren't available in every region.

- **Regional pairs**. Azure regions are grouped into regional pairs that consist of two regions in a single geography. Some Azure services use paired regions to ensure business continuity and to provide a level of protection against data loss. For example, Azure geo-redundant storage (GRS) replicates data to a secondary paired region automatically, ensuring that data is durable if the primary region isn't recoverable. If an outage affects multiple Azure regions, at least one region in each pair is prioritized for recovery.

- **Data consistency**. For consistency challenges, consider using a globally distributed data store, a stamped regional architecture, and a partially active/active deployment. In a partial deployment, some components are active across all regions while others are located centrally within the primary region.

- **Safe deployment**. The Azure safe deployment practice (SDP) framework ensures that all code and configuration changes (planned maintenance) to the Azure platform undergo a phased rollout. Health is analyzed for degradation during the release. After canary and pilot phases complete successfully, platform updates are serialized across regional pairs, so only one region in each pair is updated at a given time.

- **Platform capacity**. Like any cloud provider, Azure has finite resources. Unavailability can be the result of capacity limitations in regions. If there's a regional outage, there's an increase in demand for resources as the workload attempts to recover within the paired region. The outage might create a capacity problem, where supply temporarily doesn't meet demand.

Design recommendations

- Deploy your solution in at least two Azure regions to help protect against regional outages. Deploy it in regions that have the capabilities and characteristics that the workload requires. The capabilities should meet performance and availability targets while fulfilling data residency and retention requirements.

  For example, some data compliance requirements might constrain the number of available regions and potentially force design compromises. In such cases, we strongly recommend that you add extra investment in operational wrappers to predict, detect, and respond to failures. Suppose you're constrained to a geography with two regions, and only one of those regions supports availability zones (3 + 1 datacenter model). Create a secondary deployment pattern using fault domain isolation to allow both regions to be deployed in an active configuration, and ensure that the primary region houses multiple deployment stamps.

If suitable Azure regions don't all offer capabilities that you need, be prepared to compromise on the consistency of regional deployment stamps to prioritize geographical distribution and maximize reliability. If only a single Azure region is suitable, deploy multiple deployment stamps (regional scale units) in the selected region to mitigate some risk, and use availability zones to provide datacenter-level fault tolerance. However, such a significant compromise in geographical distribution dramatically constrains the attainable composite SLA and overall reliability.

> ⓘ **Important**
>
> For scenarios that target an SLO that's greater than or equal to 99.99%, we recommend a minimum of three deployment regions to maximize the composite SLA and overall reliability. Calculate the **composite SLA** for all user flows. Ensure that the composite SLA is aligned with business targets.

- For high-scale application scenarios that have significant volumes of traffic, design the solution to scale across multiple regions to navigate potential capacity constraints within a single region. Additional regional deployment stamps will achieve a higher composite SLA. Using global resources constrains the increase in composite SLA that you achieve by adding more regions.

- Define and validate your recovery point objectives (RPO) and recovery time objectives (RTO).

- Within a single geography, prioritize the use of regional pairs to benefit from SDP serialized rollouts for planned maintenance and regional prioritization for unplanned maintenance.

- Geographically colocate Azure resources with users to minimize network latency and maximize end-to-end performance.
  - You can also use solutions like a Content Delivery Network (CDN) or edge caching to drive optimal network latency for distributed user bases. For more information, see Global traffic routing, Application delivery services, and Caching and static content delivery.

- Align current service availability with product roadmaps when you choose deployment regions. Some services might not be immediately available in every region.

Containerization

A container includes application code and the related configuration files, libraries, and dependencies that the application needs to run. Containerization provides an abstraction layer for application code and its dependencies and creates separation from the underlying hosting platform. The single software package is highly portable and can run consistently across various infrastructure platforms and cloud providers. Developers don't need to rewrite code and can deploy applications faster and more reliably.

> ⓘ **Important**
>
> We recommend that you use containers for mission-critical application packages. They improve infrastructure utilization because you can host multiple containers on the same virtualized infrastructure. Also, because all software is included in the container, you can move the application across various operating systems, regardless of runtimes or library versions. Management is also easier with containers than it is with traditional virtualized hosting.
>
> Mission-critical applications need to scale fast to avoid performance bottlenecks. Because container images are pre-built, you can limit startup to occur only during bootstrapping of the application, which provides rapid scalability.

Design considerations

- **Monitoring**. It can be difficult for monitoring services to access applications that are in containers. You typically need third-party software to collect and store container state indicators like CPU or RAM usage.
- **Security**. The hosting platform OS kernel is shared across multiple containers, creating a single point of attack. However, the risk of host VM access is limited because containers are isolated from the underlying operating system.
- **State**. Although it's possible to store data in a running container's file system, the data won't persist when the container is re-created. Instead, persist data by mounting external storage or using an external database.

Design recommendations

- Containerize all application components. Use container images as the primary model for application deployment packages.
- Prioritize Linux-based container runtimes when possible. The images are more lightweight, and new features for Linux nodes/containers are released frequently.
- Make containers immutable and replaceable, with short lifecycles.
- Be sure to gather all relevant logs and metrics from the container, container host, and underlying cluster. Send the gathered logs and metrics to a unified data sink for further processing and analysis.
- Store container images in Azure Container Registry. Use geo-replication to replicate container images across all regions. Enable Microsoft Defender for container registries to provide vulnerability scanning for container images. Make sure access to the registry is managed by Azure Active Directory (Azure AD).

Container hosting and orchestration

Several Azure application platforms can effectively host containers. There are advantages and disadvantages associated with each of these platforms. Compare the options in the context of your business requirements. However, always optimize reliability, scalability, and performance. For more information, see these articles:

- Compute decision tree
- Container option comparisons

> **Important**
>
> **Azure Kubernetes Service (AKS)** should be your first choice of orchestrator when it meets your requirements. **Azure Container Apps** is another option. Although **Azure App Service** isn't an orchestrator, as a low-friction container platform, it's still a feasible alternative to AKS.

Design considerations and recommendations for Azure Kubernetes Service

AKS, a managed Kubernetes service, enables quick cluster provisioning without requiring complex cluster administration activities and offers a feature set that includes advanced networking and identity capabilities. For a complete set of recommendations, see Azure Well-Architected Framework review - AKS.

> ⓘ **Important**
>
> There are some foundational configuration decisions that you can't change without re-deploying the AKS cluster. Examples include the choice between public and private AKS clusters, enabling Azure Network Policy, Azure AD integration, and the use of managed identities for AKS instead of service principals.

Reliability

AKS manages the native Kubernetes control plane. If the control plane isn't available, the workload experiences downtime. Take advantage of the reliability features offered by AKS:

- Deploy AKS clusters across different Azure regions as a scale unit to maximize reliability and availability. Use availability zones to maximize resilience within an Azure region by distributing AKS control plane and agent nodes across physically separate datacenters. However, if colocation latency is a problem, you can do AKS deployment within a single zone or use proximity placement groups to minimize internode latency.

- Use the AKS Uptime SLA for production clusters to maximize Kubernetes API endpoint availability guarantees.

Scalability

Take into account AKS scale limits, like the number of nodes, node pools per cluster, and clusters per subscription.

- If scale limits are a constraint, take advantage of the scale-unit strategy, and deploy more units with clusters.

- Enable cluster autoscaler to automatically adjust the number of agent nodes in response to resource constraints.

- Use the horizontal pod autoscaler to adjust the number of pods in a deployment based on CPU utilization or other metrics.

- For high scale and burst scenarios, consider using virtual nodes for extensive and rapid scale.

- Define pod resource requests and limits in application deployment manifests. If you don't, you might experience performance problems.

Isolation

Maintain boundaries between the infrastructure used by the workload and system tools. Sharing infrastructure might lead to high-resource utilization and noisy neighbor scenarios.

- Use separate node pools for system and workload services. Dedicated node pools for workload components should be based on requirements for specialized infrastructure resources like high-memory GPU VMs. In general, to reduce unnecessary management overhead, avoid deploying large numbers of node pools.
- Use taints and tolerations to provide dedicated nodes and limit resource-intensive applications.
- Evaluate application affinity and anti-affinity requirements and configure the appropriate colocation of containers on nodes.

Security

Default vanilla Kubernetes requires significant configuration to ensure a suitable security posture for mission-critical scenarios. AKS addresses various security risks out of the box. Features include private clusters, auditing and logging into Log Analytics, hardened node images, and managed identities.

- Apply configuration guidance provided in the AKS security baseline.
- Use AKS features for handling cluster identity and access management to reduce operational overhead and apply consistent access management.
- Use managed identities instead of service principals to avoid management and rotation of credentials. You can add managed identities at the cluster level. At the pod level, you can use managed identities via Azure AD workload identity.
- Use Azure AD integration for centralized account management and passwords, application access management, and enhanced identity protection. Use Kubernetes RBAC with Azure AD for least privilege, and minimize granting administrator privileges to help protect configuration and secrets access. Also, limit access to the Kubernetes cluster configuration file by using Azure role-based

access control. Limit access to actions that containers can perform, provide the least number of permissions, and avoid the use of root privilege escalation.

Upgrades

Clusters and nodes need to be upgraded regularly. AKS supports Kubernetes versions in alignment with the release cycle of native Kubernetes.

- Subscribe to the public AKS Roadmap and Release Notes on GitHub to stay up-to-date on upcoming changes, improvements, and, most importantly, Kubernetes version releases and deprecations.

- Apply the guidance provided in the AKS checklist to ensure alignment with best practices.

- Be aware of the various methods supported by AKS for updating nodes and/or clusters. These methods can be manual or automated. You can use Planned Maintenance to define maintenance windows for these operations. New images are released weekly. AKS also supports auto-upgrade channels for automatically upgrading AKS clusters to newer versions of Kubernetes and/or newer node images when they're available.

Networking

Evaluate the network plugins that best fit your use case. Determine whether you need granular control of traffic between pods. Azure supports kubenet, Azure CNI, and bring your own CNI for specific use cases.

Prioritize the use of Azure CNI after assessing network requirements and the size of the cluster. Azure CNI enables the use of Azure or Calico network policies for controlling traffic within the cluster.

Monitoring

Your monitoring tools should be able to capture logs and metrics from running pods. You should also gather information from the Kubernetes Metrics API to monitor the health of running resources and workloads.

- Use Azure Monitor and Application Insights to collect metrics, logs, and diagnostics from AKS resources for troubleshooting.

- Enable and review Kubernetes resource logs.

- Configure Prometheus metrics in Azure Monitor. Container insights in Monitor provides onboarding, enables monitoring capabilities out of the box, and enables more advanced capabilities via built-in Prometheus support.

Governance

Use policies to apply centralized safeguards to AKS clusters in a consistent way. Apply policy assignments at a subscription scope or higher to drive consistency across development teams.

- Control which functions are granted to pods, and whether running contradicts policy, by using Azure Policy. This access is defined through built-in policies provided by the Azure Policy Add-on for AKS.
- Establish a consistent reliability and security baseline for AKS cluster and pod configurations by using Azure Policy.
- Use the Azure Policy Add-on for AKS to control pod functions, like root privileges, and to disallow pods that don't conform to policy.

> ⓘ **Note**
>
> When you deploy into an Azure landing zone, the Azure policies to help you ensure consistent reliability and security should be provided by the landing zone implementation.
>
> The mission-critical **reference implementations** provide a suite of baseline policies to drive recommended reliability and security configurations.

Design considerations and recommendations for Azure App Service

For web and API-based workload scenarios, App Service might be a feasible alternative to AKS. It provides a low-friction container platform without the complexity of Kubernetes. For a complete set of recommendations, see Reliability considerations for App Service and Operational excellence for App Service.

Reliability

Evaluate the use of TCP and SNAT ports. TCP connections are used for all outbound connections. SNAT ports are used for outbound connections to public IP addresses. SNAT port exhaustion is a common failure scenario. You should predictively detect this problem by load testing while using Azure Diagnostics to monitor ports. If SNAT errors occur, you need to either scale across more or larger workers or implement coding practices to help preserve and reuse SNAT ports. Examples of coding practices that you can use include connection pooling and the lazy loading of resources.

TCP port exhaustion is another failure scenario. It occurs when the sum of outbound connections from a given worker exceeds capacity. The number of available TCP ports depends on the size of the worker. For recommendations, see TCP and SNAT ports.

Scalability

Plan for future scalability requirements and application growth so that you can apply appropriate recommendations from the start. By doing so, you can avoid technical migration debt as the solution grows.

- Enable autoscale to ensure that adequate resources are available to service requests. Evaluate per-app scaling for high-density hosting on App Service.
- Be aware that App Service has a default, soft limit of instances per App Service plan.
- Apply autoscale rules. An App Service plan scales out if any rule within the profile is met but only scales in if all rules within the profile are met. Use a scale-out and scale-in rule combination to ensure that autoscale can take action to both scale out and scale in. Understand the behavior of multiple scaling rules in a single profile.
- Be aware that you can enable per-app scaling at the level of the App Service plan to allow an application to scale independently from the App Service plan that hosts it. Apps are allocated to available nodes via a best-effort approach for an even distribution. Although an even distribution isn't guaranteed, the platform ensures that two instances of the same app aren't hosted on the same instance.

Monitoring

Monitor application behavior and get access to relevant logs and metrics to ensure that your application works as expected.

- You can use diagnostic logging to ingest application-level and platform-level logs into Log Analytics, Azure Storage, or a third-party tool via Azure Event Hubs.
- Application performance monitoring with Application Insights provides deep insights into application performance.

- Mission-critical applications must have the ability to self-heal if there are failures. Enable Auto Heal to automatically recycle unhealthy workers.

- You need to use appropriate health checks to assess all critical downstream dependencies, which helps to ensure overall health. We strongly recommend that you enable Health Check to identify non-responsive workers.

Deployment

To work around the default limit of instances per App Service plan, deploy App Service plans in multiple scale units in a single region. Deploy App Service plans in an availability zone configuration to ensure that worker nodes are distributed across zones within a region. Consider opening a support ticket to increase the maximum number of workers to twice the instance count that you need to serve normal peak load.

Container registry

Container registries host images that are deployed to container runtime environments like AKS. You need to configure your container registries for mission-critical workloads carefully. An outage shouldn't cause delays in pulling images, especially during scaling operations. The following considerations and recommendations focus on Azure Container Registry and explore the trade-offs that are associated with centralized and federated deployment models.

Design considerations

- **Format**. Consider using a container registry that relies on the Docker-provided format and standards for both push and pull operations. These solutions are compatible and mostly interchangeable.

- **Deployment model**. You can deploy the container registry as a centralized service that's consumed by multiple applications within your organization. Or you can deploy it as a dedicated component for a specific application workload.

- **Public registries**. Container images are stored in Docker Hub or other public registries that exist outside of Azure and a given virtual network. This isn't necessarily a problem, but it can lead to various issues that are related to service availability, throttling, and data exfiltration. For some application scenarios, you need to replicate public container images in a private container registry to limit egress traffic, increase availability, or avoid potential throttling.

Design recommendations

- Use container registry instances that are dedicated to the application workload. Avoid creating a dependency on a centralized service unless organizational availability and reliability requirements are fully aligned with the application.

  In the recommended core architecture pattern, container registries are global resources that are long living. Consider using a single global container registry per environment. For example, use a global production registry.

- Ensure that the SLA for public registry is aligned with your reliability and security targets. Take special note of throttling limits for use cases that depend on Docker Hub.

- Prioritize Azure Container Registry for hosting container images.

Design considerations and recommendations for Azure Container Registry

This native service provides a range of features, including geo-replication, Azure AD authentication, automated container building, and patching via Container Registry tasks.

Reliability

Configure geo-replication to all deployment regions to remove regional dependencies and optimize latency. Container Registry supports high availability through geo-replication to multiple configured regions, providing resiliency against regional outages. If a region becomes unavailable, the other regions continue to serve image requests. When the region is back online, Container Registry recovers and replicates changes to it. This capability also provides registry colocation within each configured region, reducing network latency and cross-region data transfer costs.

In Azure regions that provide availability zone support, the Premium Container Registry tier supports zone redundancy to provide protection against zonal failure. The Premium tier also supports private endpoints to help prevent unauthorized access to the registry, which can lead to reliability issues.

Host images close to the consuming compute resources, within the same Azure regions.

Image locking

Images can get deleted, as a result of, for example, manual error. Container Registry supports locking an image version or a repository to prevent changes or deletions.

When a previously deployed image *version* is changed in place, same-version deployments might provide different results before and after the change.

If you want to protect the Container Registry instance from deletion, use resource locks.

Tagged images

Tagged Container Registry images are mutable by default, which means that the same tag can be used on multiple images pushed to the registry. In production scenarios, this can lead to unpredictable behavior that could affect application uptime.

Identity and access management

Use Azure AD integrated authentication to push and pull images instead of relying on access keys. For enhanced security, fully disable the use of the admin access key.

Serverless compute

Serverless computing provides resources on demand and eliminates the need to manage infrastructure. The cloud provider automatically provisions, scales, and manages the resources required to run deployed application code. Azure provides several serverless compute platforms:

- Azure Functions. When you use Azure Functions, application logic is implemented as distinct blocks of code, or *functions*, that run in response to events, like an HTTP request or queue message. Each function scales as necessary to meet demand.

- Azure Logic Apps. Logic Apps is best suited for creating and running automated workflows that integrate various apps, data sources, services, and systems. Like Azure Functions, Logic Apps uses built-in triggers for event-driven processing. However, instead of deploying application code, you can create logic apps by using a graphical user interface that supports code blocks like conditionals and loops.

- Azure API Management. You can use API Management to publish, transform, maintain, and monitor enhanced-security APIs by using the Consumption tier.

- Power Apps and Power Automate. These tools provide a low-code or no-code development experience, with simple workflow logic and integrations that are configurable through connections in a user interface.

For mission-critical applications, serverless technologies provide simplified development and operations, which can be valuable for simple business use cases. However, this simplicity comes at the cost of flexibility in terms of scalability, reliability, and performance, and that's not viable for most mission-critical application scenarios.

The following sections provide design considerations and recommendations for using Azure Functions and Logic Apps as alternative platforms for non-critical workflow scenarios.

Design considerations and recommendations for Azure Functions

Mission-critical workloads have critical and non-critical system flows. Azure Functions is a viable choice for flows that don't have the same stringent business requirements as critical system flows. It's well suited for event-driven flows that have short-lived processes because functions perform distinct operations that run as fast as possible.

Choose an Azure Functions hosting option that's appropriate for the application's reliability tier. We recommend the Premium plan because it allows you to configure compute instance size. The Dedicated plan is the least serverless option. It provides autoscale, but these scale operations are slower than those of the other plans. We recommend that you use the Premium plan to maximize reliability and performance.

There are some security considerations. When you use an HTTP trigger to expose an external endpoint, use a web application firewall (WAF) to provide a level of protection for the HTTP endpoint from common external attack vectors.

We recommend the use of private endpoints for restricting access to private virtual networks. They can also mitigate data exfiltration risks, like malicious admin scenarios.

You need to use code scanning tools on Azure Functions code and integrate those tools with CI/CD pipelines.

Design considerations and recommendations for Azure Logic Apps

Like Azure Functions, Logic Apps uses built-in triggers for event-driven processing. However, instead of deploying application code, you can create logic apps by using a graphical user interface that supports blocks like conditionals, loops, and other constructs.

Multiple deployment modes are available. We recommend the Standard mode to ensure a single-tenant deployment and mitigate noisy neighbor scenarios. This mode uses the containerized single-tenant Logic Apps runtime, which is based on Azure Functions. In this mode, the logic app can have multiple stateful and stateless workflows. You should be aware of the configuration limits.

Constrained migrations via IaaS

Many applications that have existing on-premises deployments use virtualization technologies and redundant hardware to provide mission-critical levels of reliability. Modernization is often hindered by business constraints that prevent full alignment with the cloud-native baseline (North Star) architecture pattern that's recommended for mission-critical workloads. That's why many applications adopt a phased approach, with initial cloud deployments using virtualization and Azure Virtual Machines as the primary application hosting model. The use of IaaS virtual machines might be required in certain scenarios:

- Available PaaS services don't provide the required performance or level of control.
- The workload requires operating system access, specific drivers, or network and system configurations.
- The workload doesn't support running in containers.
- There's no vendor support for third-party workloads.

This section focuses on the best ways to use Azure Virtual Machines and associated services to maximize the reliability of the application platform. It highlights key aspects of the mission-critical design methodology that transpose cloud-native and IaaS migration scenarios.

Design considerations

- The operational costs of using IaaS virtual machines are significantly higher than the costs of using PaaS services because of the management requirements of the virtual machines and the operating systems. Managing virtual machines necessitates the frequent rollout of software packages and updates.

- Azure provides capabilities to increase the availability of virtual machines:
  - Availability sets can help protect against network, disk, and power failures by distributing virtual machines across fault domains and update domains.
  - Availability zones can help you achieve even higher levels of reliability by distributing VMs across physically separated datacenters within a region.
  - Virtual Machine Scale Sets provide functionality for automatically scaling the number of virtual machines in a group. They also provide capabilities for monitoring instance health and automatically repairing unhealthy instances.

Design recommendations

ⓘ **Important**

Use PaaS services and containers when possible to reduce operational complexity and cost. Use IaaS virtual machines only when you need to.

- Right-size VM SKU sizes to ensure effective resource utilization.
- Deploy three or more virtual machines across availability zones to achieve datacenter-level fault tolerance.
  - If you're deploying commercial off-the-shelf software, consult the software vendor and test adequately before deploying the software into production.
- For workloads that can't be deployed across availability zones, use availability sets that contain three or more VMs.
  - Consider availability sets only if availability zones don't meet workload requirements, such as for chatty workloads with low latency requirements.
- Prioritize the use of Virtual Machine Scale Sets for scalability and zone redundancy. This point is particularly important for workloads that have varying loads. For example, if the number of active users or requests per second is a varying load.
- Don't access individual virtual machines directly. Use load balancers in front of them when possible.
- To protect against regional outages, deploy application virtual machines across multiple Azure regions.
  - See the networking and connectivity design area for details about how to optimally route traffic between active deployment regions.
- For workloads that don't support multi-region active/active deployments, consider implementing active/passive deployments by using hot/warm standby virtual machines for regional failover.
- Use standard images from Azure Marketplace rather than custom images that need to be maintained.
- Implement automated processes to deploy and roll out changes to virtual machines, avoiding any manual intervention. For more information, see IaaS considerations in the Operational procedures design area.
- Implement chaos experiments to inject application faults into virtual machine components, and observe the mitigation of faults. For more information, see Continuous validation and testing.
- Monitor virtual machines and ensure that diagnostic logs and metrics are ingested into a unified data sink.

- Implement security practices for mission-critical application scenarios, when applicable, and the Security best practices for IaaS workloads in Azure.

Next step

Review the considerations for the data platform.

Data platform

Data platform considerations for mission-critical workloads on Azure

Article • 02/01/2023

The selection of an effective application data platform is a further crucial decision area, which has far-reaching implications across other design areas. Azure ultimately offers a multitude of relational, non-relational, and analytical data platforms, which differ greatly in capability. It's therefore essential that key non-functional requirements be fully considered alongside other decision factors such as consistency, operability, cost, and complexity. For example, the ability to operate in a multi-region write configuration will have a critical bearing on suitability for a globally available platform.

This design area expands on application design, providing key considerations and recommendations to inform the selection of an optimal data platform.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**

The Four Vs of Big Data

The 'Four Vs of Big Data' provide a framework to better understand requisite characteristics for a highly available data platform, and how data can be used to maximize business value. This section will therefore explore how the Volume, Velocity, Variety, and Veracity characteristics can be applied at a conceptual level to help design a data platform using appropriate data technologies.

- **V**olume: how much data is coming in to inform storage capacity and tiering requirements -that is the size of the dataset.
- **V**elocity: the speed at which data is processed, either as batches or continuous streams -that is the rate of flow.
- **V**ariety: the organization and format of data, capturing structured, semi-structured, and unstructured formats -that is data across multiple stores or types.
- **V**eracity: includes the provenance and curation of considered data sets for governance and data quality assurance -that is accuracy of the data.

Design Considerations

Volume

- Existing (if any) and expected future data volumes based on forecasted data growth rates aligned with business objectives and plans.
  - Data volume should encompass the data itself and indexes, logs, telemetry, and other applicable datasets.
  - Large business-critical and mission-critical applications typically generate and store high volumes (GB and TB) on a daily basis.
  - There can be significant cost implications associated with data expansion.

- Data volume may fluctuate due to changing business circumstances or housekeeping procedures.

- Data volume can have a profound impact on data platform query performance.

- There can be a profound impact associated with reaching data platform volume limits.
  - *Will it result in downtime? and if so, for how long?*
  - *What are the mitigation procedures? and will mitigation require application changes?*
  - *Will there be a risk of data loss?*

- Features such as Time to Live (TTL) can be used to manage data growth by automatically deleting records after an elapsed time, using either record creation or modification.
  - For example, Azure Cosmos DB provides an in-built TTL capability.

Velocity

- The speed with which data is emitted from various application components, and the throughput requirements for how fast data needs to be committed and retrieved are critical to determining an optimal data technology for key workload scenarios.
  - The nature of throughput requirements will differ by workload scenario, such as those that are read-heavy or write-heavy.
    - For example, analytical workloads will typically need to cater to a large read throughput.
  - *What is the required throughput? And how is throughput expected to grow?*
  - *What are the data latency requirements at P50/P99 under reference load levels?*

- Capabilities such as supporting a lock-free design, index tuning, and consistency policies are critical to achieving high-throughput.

    - Optimizing configuration for high throughput incurs trade-offs, which should be fully understood.
    - Load-levelling persistence and messaging patterns, such as CQRS and Event Sourcing, can be used to further optimize throughput.

- Load levels will naturally fluctuate for many application scenarios, with natural peaks requiring a sufficient degree of elasticity to handle variable demand while maintaining throughput and latency.
    - Agile scalability is key to effectively support variable throughput and load levels without overprovisioning capacity levels.
        - Both read and write throughput must scale according to application requirements and load.
        - Both vertical and horizontal scale operations can be applied to respond to changing load levels.

- The impact of throughput dips can vary based on workload scenario.
    - *Will there be connectivity disruption?*
    - *Will individual operations return failure codes while the control plane continues to operate?*
    - *Will the data platform activate throttling, and if so for how long?*

- The fundamental application design recommendation to use an active-active geographical distribution introduces challenges around data consistency.
    - There's a trade-off between consistency and performance with regard to full ACID transactional semantics and traditional locking behavior.
        - Minimizing write latency will come at the cost of data consistency.

- In a multi-region write configuration, changes will need to be synchronized and merged between all replicas, with conflict resolution where required, and this may impact performance levels and scalability.

- Read-only replicas (intra-region and inter-region) can be used to minimize roundtrip latency and distributing traffic to boost performance, throughput, availability, and scalability.

- A caching layer can be used to increase read throughput to improve user experience and end-to-end client response times.
    - Cache expiration times and policies need to be considered to optimize data recentness.

Variety

- The data model, data types, data relationships, and intended query model will strongly affect data platform decisions.
  - *Does the application require a relational data model or can it support a variable-schema or non-relational data model?*
  - *How will the application query data? And will queries depend on database-layer concepts such as relational joins? Or does the application provide such semantics?*

- The nature of datasets considered by the application can be varied, from unstructured content such as images and videos, or more structured files such as CSV and Parquet.
  - Composite application workloads will typically have distinct datasets and associated requirements.

- In addition to relational or non-relational data platforms, graph or key-value data platforms may also be suitable for certain data workloads.
  - Some technologies cater to variable-schema data models, where data items are semantically similar and/or stored and queried together but differ structurally.

- In a microservice architecture, individual application services can be built with distinct scenario-optimized datastores rather than depending on a single monolithic datastore.
  - Design patterns such as SAGA can be applied to manage consistency and dependencies between different datastores.
    - Inter-database direct queries can impose co-location constraints.
  - The use of multiple data technologies will add a degree of management overhead to maintain encompassed technologies.

- The feature-sets for each Azure service differ across languages, SDKs, and APIs, which can greatly impact the level of configuration tuning that can be applied.

- Capabilities for optimized alignment with the data model and encompassed data types will strongly influence data platform decisions.
  - Query layers such as stored procedures and object-relational mappers.
  - Language-neutral query capability, such as a secured REST API layer.
  - Business continuity capabilities, such as backup and restore.

- Analytical datastores typically support polyglot storage for various types of data structures.
  - Analytical runtime environments, such as Apache Spark, may have integration restrictions to analyze polyglot data structures.

- In an enterprise context, the use of existing processes and tooling, and the continuity of skills, can have a significant bearing on the data platform design and selection of data technologies.

Veracity

- Several factors must be considered to validate the accuracy of data within an application, and the management of these factors can have a significant bearing on the design of the data platform.
  - Data consistency.
  - Platform security features.
  - Data governance.
  - Change management and schema evolution.
  - Dependencies between datasets.

- In any distributed application with multiple data replicas there's a trade-off between consistency and latency, as expressed in the CAP and PACELC theorems.
  - When readers and writers are distinctly distributed, an application must choose to return either the fastest-available version of a data item, even if it out of date compared to a just-completed write (update) of that data item in another replica, or the most up-to-date version of the data item, which may incur additional latency to determine and obtain the latest state.
  - Consistency and availability can be configured at platform level or at individual data request level.
  - *What is the user experience if data was to be served from a replica closest to the user which doesn't reflect the most recent state of a different replica? i.e. can the application support possibly serving out-of-date data?*

- In a multi-region write context, when the same data item is changed in two separate write-replicas before either change can be replicated, a conflict is created which must be resolved.
  - Standardized conflict resolution policies, such as "Last Write Wins", or a custom strategy with custom logic can be applied.

- The implementation of security requirements may adversely impact throughput or performance.

- Encryption at-rest can be implemented in the application layer using client-side encryption and/or the data layer using server-side encryption if necessary.

- Azure supports various encryption models, including server-side encryption that uses service-managed keys, customer-managed keys in Key Vault, or customer-managed keys on customer-controlled hardware.

  - With client-side encryption, keys can be managed in Key Vault or another secure location.

- MACsec (IEEE 802.1AE MAC) data-link encryption is used to secure all traffic moving between Azure datacenters on the Microsoft backbone network.
  - Packets are encrypted and decrypted on the devices before being sent, preventing physical 'man-in-the-middle' or snooping/wiretapping attacks.

- Authentication and authorization to the data plane and control plane.
  - *How will the data platform authenticate and authorize application access and operational access?*

- Observability through monitoring platform health and data access.
  - *How will alerting be applied for conditions outside acceptable operational boundaries?*

Design Recommendations

Volume

- Ensure future data volumes associated with organic growth aren't expected to exceed data platform capabilities.
  - Forecast data growth rates aligned to business plans and use established rates to inform ongoing capacity requirements.
  - Compare aggregate and per-data record volumes against data platform limits.
  - If there's a risk of limits being reached in exceptional circumstances, ensure operational mitigations are in place to prevent downtime and data loss.

- Monitor data volume and validate it against a capacity model, considering scale limits and expected data growth rates.
  - Ensure scale operations align with storage, performance, and consistency requirements.
  - When a new scale-unit's introduced, underlying data may need to be replicated which will take time and likely introduce a performance penalty while replication occurs. So ensure these operations are performed outside of critical business hours if possible.

- Define application data tiers to classify datasets based on usage and criticality to facilitate the removal or offloading of older data.
  - Consider classifying datasets into 'hot', 'warm', and 'cold' ('archive') tiers.
    - For example, the foundational reference implementations use Azure Cosmos DB to store 'hot' data that is actively used by the application, while Azure Storage is used for 'cold' operations data for analytical purposes.

- Configure housekeeping procedures to optimize data growth and drive data efficiencies, such as query performance, and managing data expansion.
  - Configure Time-To-Live (TTL) expiration for data that is no-longer required and has no long-term analytical value.
    - Validate that old data can be safely tiered to secondary storage, or deleted outright, without an adverse impact to the application.
  - Offload non-critical data to secondary cold storage, but maintain it for analytical value and to satisfy audit requirements.
  - Collect data platform telemetry and usage statistics to enable DevOps teams to continually evaluate housekeeping requirements and 'right-size' datastores.

- In-line with a microservice application design, consider the use of multiple different data technologies in-parallel, with optimized data solutions for specific workload scenarios and volume requirements.
  - Avoid creating a single monolithic datastore where data volume from expansion can be hard to manage.

Velocity

- The data platform must inherently be designed and configured to support high-throughput, with workloads separated into different contexts to maximize performance using scenario optimized data solutions.
  - Ensure read and write throughput for each data scenario can scale according to expected load patterns, with sufficient tolerance for unexpected variance.
  - Separate different data workloads, such as transactional and analytical operations, into distinct performance contexts.

- Load-level through the use of asynchronous non-blocking messaging, for example using the CQRS or Event Sourcing patterns.
  - There might be latency between write requests and when new data becomes available to read, which may have an impact on the user experience.
    - This impact must be understood and acceptable in the context of key business requirements.

- Ensure agile scalability to support variable throughput and load levels.
  - If load levels are highly volatile, consider overprovisioning capacity levels to ensure throughput and performance is maintained.
  - Test and validate the impact to composite application workloads when throughput can't be maintained.

- Prioritize Azure-native data services with automated scale-operations to facilitate a swift response to load-level volatility.
  - Configure autoscaling based on service-internal and application-set thresholds.

  - Scaling should initiate and complete in timeframes consistent with business requirements.
  - For scenarios where manual interaction is necessary, establish automated operational 'playbooks' that can be triggered rather than conducting manual operational actions.
    - Consider whether automated triggers can be applied as part of subsequent engineering investments.

- Monitor application data read and write throughput against P50/P99 latency requirements and align to an application capacity model.

- Excess throughput should be gracefully handled by the data platform or application layer and captured by the health model for operational representation.

- Implement caching for 'hot' data scenarios to minimize response times.
  - Apply appropriate policies for cache expiration and house-keeping to avoid runaway data growth.
    - Expire cache items when the backing data changes.
    - If cache expiration is strictly Time-To-Live (TTL) based, the impact and customer experience of serving outdated data needs to be understood.

Variety

- In alignment with the principle of a cloud- and Azure-native design, it's highly recommended to prioritize managed Azure services to reduce operational and management complexity, and taking advantage of Microsoft's future platform investments.

- In alignment with the application design principle of loosely coupled microservice architectures, allow individual services to use distinct data stores and scenario-optimized data technologies.
  - Identify the types of data structure the application will handle for specific workload scenarios.
  - Avoid creating a dependency on a single monolithic datastore.
    - Consider the SAGA design pattern where dependencies between datastores exist.

- Validate that required capabilities are available for selected data technologies.
  - Ensure support for required languages and SDK capabilities. Not every capability is available for every language/SDK in the same fashion.

Veracity

- Adopt a multi-region data platform design and distribute replicas across regions for maximum reliability, availability, and performance by moving data closer to application endpoints.
  - Distribute data replicas across Availability Zones (AZs) within a region (or use zone-redundant service tiers) to maximize intra-region availability.

- Where consistency requirements allow for it, use a multi-region write data platform design to maximize overall global availability and reliability.
  - Consider business requirements for conflict resolution when the same data item is changed in two separate write replicas before either change can be replicated and thus creating a conflict.
    - Use standardized conflict resolution policies such as "Last one wins" where possible
      - If a custom strategy with custom logic is required, ensure CI/CD DevOps practices are applied to manage custom logic.

- Test and validate backup and restore capabilities, and failover operations through chaos testing within continuous delivery processes.

- Run performance benchmarks to ensure throughput and performance requirements aren't impacted by the inclusion of required security capabilities, such as encryption.
  - Ensure continuous delivery processes consider load testing against known performance benchmarks.

- When applying encryption, it's strongly recommended to use service-managed encryption keys as a way of reducing management complexity.
  - If there's a specific security requirement for customer-managed keys, ensure appropriate key management procedures are applied to ensure availability, backup, and rotation of all considered keys.

> ⓘ **Note**
>
> When integrating with a broader organizational implementation, it's critical that an application centric approach be applied for the provisioning and operation of data platform components in an application design.
>
> More specifically, to maximize reliability it's critical that individual data platform components appropriately respond to application health through operational actions which may include other application components. For example, in a scenario where additional data platform resources are needed, scaling the data platform along with other application components according to a capacity model

will likely be required, potentially through the provision of additional scale units. This approach will ultimately be constrained if there's a hard dependency of a centralized operations team to address issues related to the data platform in isolation.

Ultimately, the use of centralized data services (that is Central IT DBaaS) introduces operational bottlenecks that significantly hinder agility through a largely uncontextualized management experience, and should be avoided in a mission-critical or business-critical context.

Additional references

Additional data-platform guidance is available within the Azure Application Architecture Guide.

- Azure Data Store Decision Tree
- Criteria for choosing a Data Store
- Non-Relational Data Stores
- Relational OLTP Data Stores

Globally distributed multi-region write datastore

To fully accommodate the globally distributed active-active aspirations of an application design, it's strongly recommended to consider a distributed multi-region write data platform, where changes to separate writeable replicas are synchronized and merged between all replicas, with conflict resolution where required.

ⓘ **Important**

The microservices may not all require a distributed multi-region write datastore, so consideration should be given to the architectural context and business requirements of each workload scenario.

Azure Cosmos DB provides a globally distributed and highly available NoSQL datastore, offering multi-region writes and tunable consistency out-of-the-box. The design considerations and recommendations within this section will therefore focus on optimal Azure Cosmos DB usage.

Design considerations

Azure Cosmos DB

- Azure Cosmos DB stores data within Containers, which are indexed, row-based transactional stores designed to allow fast transactional reads and writes with response times on the order of milliseconds.

- Azure Cosmos DB supports multiple different APIs with differing feature sets, such as SQL, Cassandra, and MongoDB.
  - The first-party Azure Cosmos DB for NoSQL provides the richest feature set and is typically the API where new capabilities will become available first.

- Azure Cosmos DB supports Gateway and Direct connectivity modes, where Direct facilitates connectivity over TCP to backend Azure Cosmos DB replica nodes for improved performance with fewer network hops, while Gateway provides HTTPS connectivity to frontend gateway nodes.
  - Direct mode is only available when using the Azure Cosmos DB for NoSQL and is currently only supported on .NET and Java SDK platforms.

- Within Availability Zone enabled regions, Azure Cosmos DB offers Availability Zone (AZ) redundancy support for high availability and resiliency to zonal failures within a region.

- Azure Cosmos DB maintains four replicas of data within a single region, and when Availability Zone (AZ) redundancy is enabled, Azure Cosmos DB ensures data replicas are placed across multiple AZs to protect against zonal failures.
  - The Paxos consensus protocol is applied to achieve quorum across replicas within a region.

- An Azure Cosmos DB account can easily be configured to replicate data across multiple regions to mitigate the risk of a single region becoming unavailable.
  - Replication can be configured with either single-region writes or multi-region writes.
    - With single region writes, a primary 'hub' region is used to serve all writes and if this 'hub' region becomes unavailable, a failover operation must occur to promote another region as writable.
    - With multi-region writes, applications can write to any configured deployment region, which will replicate changes between all other regions. If a region is unavailable then the remaining regions will be used to serve write traffic.

- In a multi-region write configuration, update (insert, replace, delete) conflicts can occur where writers concurrently update the same item in multiple regions.

- Azure Cosmos DB provides two conflict resolution policies, which can be applied to automatically address conflicts.
  - Last Write Wins (LWW) applies a time-synchronization clock protocol using a system-defined timestamp `_ts` property as the conflict resolution path. If of a conflict the item with the highest value for the conflict resolution path becomes the winner, and if multiple items have the same numeric value then the system selects a winner so that all regions can converge to the same version of the committed item.
    - With delete conflicts, the deleted version always wins over either insert or replace conflicts regardless of conflict resolution path value.
    - Last Write Wins is the default conflict resolution policy.
    - When using Azure Cosmos DB for NoSQL, a custom numerical property such as a custom timestamp definition can be used for conflict resolution.
  - Custom resolution policies allow for application-defined semantics to reconcile conflicts using a registered merge stored procedure that is automatically invoked when conflicts are detected.
    - The system provides exactly once guarantee for the execution of a merge procedure as part of the commitment protocol.
    - A custom conflict resolution policy is only available with Azure Cosmos DB for NoSQL and can only be set at container creation time.

- In a multi-region write configuration, there's a dependency on a single Azure Cosmos DB 'hub' region to perform all conflict resolutions, with the Paxos consensus protocol applied to achieve quorum across replicas within the hub region.
  - The platform provides a message buffer for write conflicts within the hub region to load level and provide redundancy for transient faults.
    - The buffer is capable of storing a few minutes worth of write updates requiring consensus.

> The strategic direction of the Azure Cosmos DB platform is to remove this single region dependency for conflict resolution in a multi-region write configuration, utilizing a 2-phase Paxos approach to attain quorum at a global level and within a region.

- The primary 'hub' region is determined by the first region that Azure Cosmos DB is configured within.

    - A priority ordering is configured for additional satellite deployment regions for failover purposes.

- The data model and partitioning across logical and physical partitions plays an important role in achieving optimal performance and availability.

- When deployed with a single write region, Azure Cosmos DB can be configured for automatic failover based on a defined failover priority considering all read region replicas.

- The RTO provided by the Azure Cosmos DB platform is ~10-15 minutes, capturing the elapsed time to perform a regional failover of the Azure Cosmos DB service if a catastrophic disaster impacting the hub region.
    - This RTO is also relevant in a multi-region write context given the dependency on a single 'hub' region for conflict resolution.
        - If the 'hub' region becomes unavailable, writes made to other regions will fail after the message buffer fills since conflict resolution won't be able to occur until the service fails over and a new hub region is established.

> The strategic direction of the Azure Cosmos DB platform is to reduce the RTO to ~5 minutes by allowing partition level failovers.

- Recovery Point Objectives (RPO) and Recovery Time Objectives (RTO) are configurable via consistency levels, with a trade-off between data durability and throughput.
    - Azure Cosmos DB provides a minimum RTO of 0 for a relaxed consistency level with multi-region writes or an RPO of 0 for strong consistency with single-write region.

- Azure Cosmos DB offers a 99.999% SLA for both read and write availability for Database Accounts configured with multiple Azure regions as writable.
    - The SLA is represented by the Monthly Uptime Percentage, which is calculated as 100% - Average Error Rate.
    - The Average Error Rate is defined as the sum of Error Rates for each hour in the billing month divided by the total number of hours in the billing month, where the Error Rate is the total number of Failed Requests divided by Total Requests during a given one-hour interval.

- Azure Cosmos DB offers a 99.99% SLA for throughput, consistency, availability, and latency for Database Accounts scoped to a single Azure region when configured with any of the five Consistency Levels.

    - A 99.99% SLA also applies to Database Accounts spanning multiple Azure regions configured with any of the four relaxed Consistency Levels.

- There are two types of throughput that can be provisioned in Azure Cosmos DB, standard and autoscale, which are measured using Request Units per second (RU/s).
    - Standard throughput allocates resources required to guarantee a specified RU/s value.
        - Standard is billed hourly for provisioned throughput.
    - Autoscale defines a maximum throughput value, and Azure Cosmos DB will automatically scale up or down depending on application load, between the maximum throughput value and a minimum of 10% of the maximum throughput value.
        - Autoscale is billed hourly for the maximum throughput consumed.

- Static provisioned throughput with a variable workload may result in throttling errors, which will impact perceived application availability.
    - Autoscale protects against throttling errors by enabling Azure Cosmos DB to scale up as needed, while maintaining cost protection by scaling back down when load decreases.

- When Azure Cosmos DB is replicated across multiple regions, the provisioned Request Units (RUs) are billed per region.

- There's a significant cost delta between a multi-region-write and single-region-write configuration which in many cases may make a multi-master Azure Cosmos DB data platform cost prohibitive.

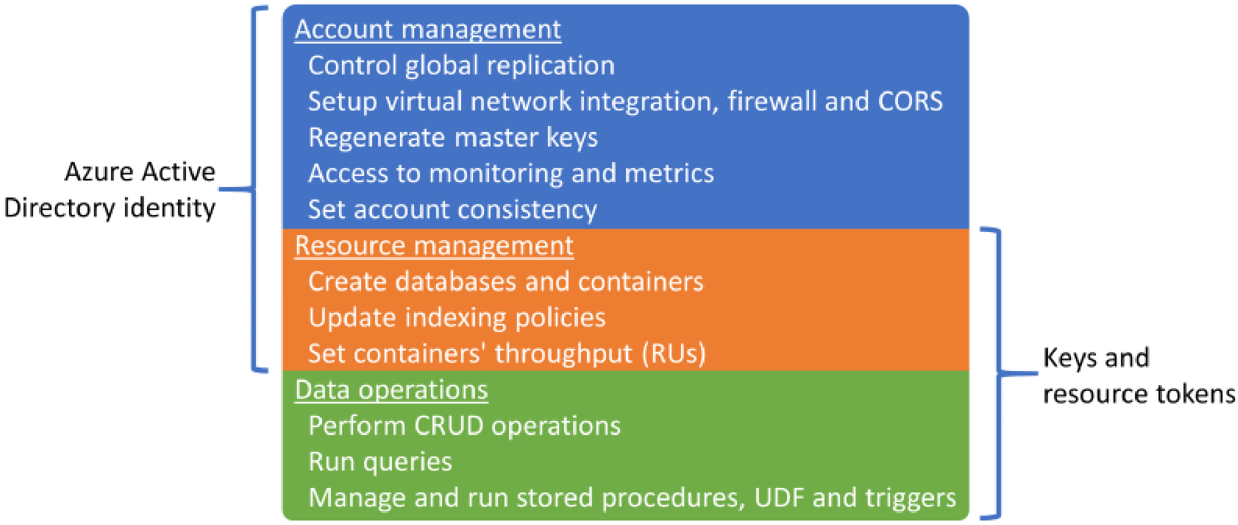
Single Region Read/Write	Single Region Write - Dual Region Read	Dual Region Read/Write
1 RU	2 RU	4 RU

> The delta between single-region-write and multi-region-write is actually less than the 1:2 ratio reflected in the table above. More specifically, there's a cross-region data transfer charge associated with write updates in a single-write configuration, which isn't captured within the RU costs as with the multi-region write configuration.

- Consumed storage is billed as a flat rate for the total amount of storage (GB) consumed to host data and indexes for a given hour.

- `Session` is the default and most widely used consistency level since data is received in the same order as writes.

- Azure Cosmos DB supports authentication via either an Azure Active Directory identity or Azure Cosmos DB keys and resource tokens, which provide overlapping capabilities.



- It's possible to disable resource management operations using keys or resource tokens to limit keys and resource tokens to data operations only, allowing for fine-grained resource access control using Azure Active Directory Role-Based Access Control (RBAC).
  - Restricting control plane access via keys or resource tokens will disable control plane operations for clients using Azure Cosmos DB SDKs and should therefore be thoroughly evaluated and tested.
  - The `disableKeyBasedMetadataWriteAccess` setting can be configured via ARM Template IaC definitions, or via a Built-In Azure Policy.

- Azure Cosmos DB Azure Active Directory RBAC support applies to account and resource control plane management operations.
  - Application administrators can create role assignments for users, groups, service principals or managed identities to grant or deny access to resources and operations on Azure Cosmos DB resources.
  - There are several Built-in RBAC Roles available for role assignment, and custom RBAC roles can also be used to form specific privilege combinations.
    - Cosmos DB Account Reader enables read-only access to the Azure Cosmos DB resource.
    - DocumentDB Account Contributor enables management of Azure Cosmos DB accounts including keys and role assignments, but doesn't enable data-plane access.

    - Cosmos DB Operator, which is similar to DocumentDB Account Contributor, but doesn't provide the ability to manage keys or role assignments.

- Azure Cosmos DB resources (accounts, databases, and containers) can be protected against incorrect modification or deletion using Resource Locks.
  - Resource Locks can be set at the account, database, or container level.
  - A Resource Lock set at on a resource will be inherited by all child resources. For example, a Resource Lock set on the Azure Cosmos DB account will be inherited by all databases and containers within the account.
  - Resource Locks **only** apply to control plane operations and do **not** prevent data plane operations, such as creating, changing, or deleting data.
  - If control plane access isn't restricted with `disableKeyBasedMetadataWriteAccess`, then clients will be able to perform control plane operations using account keys.

- The Azure Cosmos DB change feed provides a time-ordered feed of changes to data in an Azure Cosmos DB container.
  - The change feed only includes insert and update operations to the source Azure Cosmos DB container; it doesn't include deletes.

- The change feed can be used to maintain a separate data store from the primary Container used by the application, with ongoing updates to the target data store fed by the change feed from the source Container.
  - The change feed can be used to populate a secondary store for additional data platform redundancy or for subsequent analytical scenarios.

- If delete operations routinely affect the data within the source Container, then the store fed by the change feed will be inaccurate and unreflective of deleted data.
  - A Soft Delete pattern can be implemented so that data records are included in the change feed.
    - Instead of explicitly deleting data records, data records are *updated* by setting a flag (e.g. `IsDeleted`) to indicate that the item is considered deleted.
    - Any target data store fed by the change feed will need to detect and process items with a deleted flag set to True; instead of storing soft-deleted data records, the *existing* version of the data record in the target store will need to be deleted.
  - A short Time-To-Live (TTL) is typically used with the soft-delete pattern so that Azure Cosmos DB automatically deletes expired data, but only after it's reflected within the change feed with the deleted flag set to True.
    - Accomplishes the original delete intent whilst also propagating the delete through the change feed.

- Azure Cosmos DB can be configured as an analytical store, which applies a column format for optimized analytical queries to address the complexity and latency challenges that occur with the traditional ETL pipelines.

- Azure Cosmos DB automatically backs up data at regular intervals without affecting the performance or availability, and without consuming RU/s.

- Azure Cosmos DB can be configured according to two distinct backup modes.
  - Periodic is the default backup mode for all accounts, where backups are taken at a periodic interval and the data is restored by creating a request with the support team.
    - The default periodic backup retention period is 8 hours and the default backup interval is fourhours, which means only the latest two backups are stored by default.
    - The backup interval and retention period are configurable within the account.
      - The maximum retention period extends to a month with a minimum backup interval of one hour.
      - A role assignment to the Azure "Cosmos DB Account Reader Role" is required to configure backup storage redundancy.
    - Two backup copies are included at no extra cost, but additional backups incur additional costs.
    - By default, periodic backups are stored within separate Geo-Redundant Storage (GRS) that isn't directly accessible.
      - Backup storage exists within the primary 'hub' region and is replicated to the paired region through underlying storage replication.
      - The redundancy configuration of the underlying backup storage account is configurable to Zone-Redundant Storage or Locally-Redundant Storage.
    - Performing a **restore operation requires a Support Request** since customers can't directly perform a restore.
      - Before opening a support ticket, the backup retention period should be increased to at least seven days within eight hours of the data loss event.
    - A restore operation creates a new Azure Cosmos DB account where data is recovered.
      - An existing Azure Cosmos DB account can't be used for Restore
      - By default, a new Azure Cosmos DB account named `<Azure_Cosmos_account_original_name>-restored<n>` will be used.
        - This name can be adjusted, such as by reusing the existing name if the original account was deleted.
    - If throughput is provisioned at the database level, backup and restore will happen at the database level
      - It's not possible to select a subset of containers to restore.

- Continuous backup mode allows for a restore to any point of time within the last 30 days.
  - Restore operations can be performed to return to a specific point in time (PITR) with a one-second granularity.
  - The available window for restore operations is up to 30 days.
    - It's also possible to restore to the resource instantiation state.
  - Continuous backups are taken within every Azure region where the Azure Cosmos DB account exists.
    - Continuous backups are stored within the same Azure region as each Azure Cosmos DB replica, using Locally-Redundant Storage (LRS) or Zone Redundant Storage (ZRS) within regions that support Availability Zones.
  - A self-service restore can be performed using the Azure portal or IaC artifacts such as ARM templates.
  - There are several limitations with Continuous Backup.
    - The continuous backup mode isn't currently available in a multi-region-write configuration.
    - Only Azure Cosmos DB for NoSQL and Azure Cosmos DB for MongoDB can be configured for Continuous backup at this time.
    - If a container has TTL configured, restored data that has exceeded its TTL may be *immediately deleted*
  - A restore operation creates a new Azure Cosmos DB account for the point-in-time restore.
  - There's an additional storage cost for Continuous backups and restore operations.

- Existing Azure Cosmos DB accounts can be migrated from Periodic to Continuous, but not from Continuous to Periodic; migration is one-way and not reversible.

- Each Azure Cosmos DB backup is composed of the data itself and configuration details for provisioned throughput, indexing policies, deployment region(s), and container TTL settings.
  - Backups don't contain firewall settings, virtual network access control lists, private endpoint settings, consistency settings (an account is restored with session consistency), stored procedures, triggers, UDFs, or multi-region settings.
    - Customers are responsible for redeploying capabilities and configuration settings. These aren't restored via Azure Cosmos DB backup.
  - Azure Synapse Link analytical store data is also not included in Azure Cosmos DB backups.

- It's possible to implement a custom backup and restore capability for scenarios where Periodic and Continuous approaches aren't a good fit.

    - A custom approach introduces significant costs and additional administrative overhead, which should be understood and carefully assessed.
        - Common restore scenarios should be modeled, such as the corruption or deletion of an account, database, container, on data item.
        - Housekeeping procedures should be implemented to prevent backup sprawl.
    - Azure Storage or an alternative data technology can be used, such an alternative Azure Cosmos DB container.
        - Azure Storage and Azure Cosmos DB provide native integrations with Azure services such as Azure Functions and Azure Data Factory.

- The Azure Cosmos DB documentation denotes two potential options for implementing custom backups.
    - Azure Cosmos DB change feed to write data to a separate storage facility.
        - An Azure function or equivalent application process uses the change feed processor to bind to the change feed and process items into storage.
    - Both continuous or periodic (batched) custom backups can be implemented using the change feed.
    - The Azure Cosmos DB change feed doesn't yet reflect deletes, so a soft-delete pattern must be applied using a boolean property and TTL.
        - This pattern won't be required when the change feed provides full-fidelity updates.
    - Azure Data Factory Connector for Azure Cosmos DB (Azure Cosmos DB for NoSQL or MongoDB API connectors) to copy data.
        - Azure Data Factory (ADF) supports manual execution and Schedule, Tumbling window, and Event-based triggers.
            - Provides support for both Storage and Event Grid.
        - ADF is primarily suitable for periodic custom backup implementations due to its batch-oriented orchestration.
            - It's less suitable for continuous backup implementations with frequent events due to the orchestration execution overhead.
        - ADF supports Azure Private Link for high network security scenarios

> Azure Cosmos DB is used within the design of many Azure services, so a significant regional outage for Azure Cosmos DB will have a cascading effect across various Azure services within that region. The precise impact to a particular service will heavily depend on how the underlying service design uses Azure Cosmos DB.

Design Recommendations

Azure Cosmos DB

- Use Azure Cosmos DB as the primary data platform where requirements allow.

- For mission-critical workload scenarios, configure Azure Cosmos DB with a write replica inside each deployment region to reduce latency and provide maximum redundancy.
  - Configure the application to prioritize the use of the local Azure Cosmos DB replica for writes and reads to optimize application load, performance, and regional RU/s consumption.
  - The multi-region-write configuration comes at a significant cost and should be prioritized only for workload scenarios requiring maximum reliability.

- For less-critical workload scenarios, prioritize the use of single-region-write configuration (when using Availability Zones) with globally distributed read replicas, since this offers a high level of data platform reliability (99.999% SLA for read-, 99.995% SLA for write-operations) at a more compelling price-point.
  - Configure the application to use the local Azure Cosmos DB read replica to optimize read performance.

- Select an optimal 'hub' deployment region where conflict resolution will occur in a multi-region-write configuration, and all writes will be performed in a single-region-write configuration.
  - Consider distance relative to other deployment regions and associated latency in selecting a primary region, and requisite capabilities such as Availability Zones support.

- Configure Azure Cosmos DB with Availability Zone (AZ) redundancy in all deployment regions with AZ support, to ensure resiliency to zonal failures within a region.

- Use Azure Cosmos DB for NoSQL since it offers the most comprehensive feature set, particularly where performance tuning is concerned.
  - Alternative APIs should primarily be considered for migration or compatibility scenarios.
    - When using alternative APIs, validate that required capabilities are available with the selected language and SDK to ensure optimal configuration and performance.

- Use the Direct connection mode to optimize network performance through direct TCP connectivity to backend Azure Cosmos DB nodes, with a reduced number of network 'hops'.

> The Azure Cosmos DB SLA is calculated by averaging failed requests, which may not directly align with a 99.999% reliability tier error budget. When designing for

> 99.999% SLO, it's therefore vital to plan for regional and multi-region Azure Cosmos DB write unavailability, ensuring a fallback storage technology is positioned if a failure, such as a persisted message queue for subsequent replay.

- Define a partitioning strategy across both logical and physical partitions to optimize data distribution according to the data model.
  - Minimize cross-partition queries.
  - Iteratively test and validate the partitioning strategy to ensure optimal performance.
- Select an optimal partition key.
  - The partition key can't be changed after it has been created within the collection.
  - The partition key should be a property value that doesn't change.
  - Select a partition key that has a high cardinality, with a wide range of possible values.
  - The partition key should spread RU consumption and data storage evenly across all logical partitions to ensure even RU consumption and storage distribution across physical partitions.
  - Run read queries against the partitioned column to reduce RU consumption and latency.
- Indexing is also crucial for performance, so ensure index exclusions are used to reduce RU/s and storage requirements.
  - Only index those fields that are needed for filtering within queries; design indexes for the most-used predicates.
- Leverage the built-in error handling, retry, and broader reliability capabilities of the Azure Cosmos DB SDK.
  - Implement retry logic within the SDK on clients.
- Use service-managed encryption keys to reduce management complexity.
  - If there's a specific security requirement for customer-managed keys, ensure appropriate key management procedures are applied, such as backup and rotation.
- Disable Azure Cosmos DB key-based metadata write access by applying the built-in Azure Policy.
- Enable Azure Monitor to gather key metrics and diagnostic logs, such as provisioned throughput (RU/s).

  - Route Azure Monitor operational data into a Log Analytics workspace dedicated to Azure Cosmos DB and other global resources within the application design.
  - Use Azure Monitor metrics to determine if application traffic patterns are suitable for autoscale.

- Evaluate application traffic patterns to select an optimal option for provisioned throughput types.
  - Consider auto-scale provisioned throughput to automatically level-out workload demand.

- Evaluate Microsoft performance tips for Azure Cosmos DB to optimize client-side and server-side configuration for improved latency and throughput.

- When using AKS as the compute platform: For query-intensive workloads, select an AKS node SKU that has accelerated networking enabled to reduce latency and CPU jitters.

- For single write region deployments, it's strongly recommended to configure Azure Cosmos DB for automatic failover.

- Load-level through the use of asynchronous non-blocking messaging within system flows, which write updates to Azure Cosmos DB.
  - Consider patterns such as Command and Query Responsibility Segregation and Event Sourcing.

- Configure the Azure Cosmos DB account for continuous backups to obtain a fine granularity of recovery points across the last 30 days.
  - Consider the use of Azure Cosmos DB backups in scenarios where contained data or the Azure Cosmos DB account is deleted or corrupted.
  - Avoid the use of a custom backup approach unless absolutely necessary.

- It's strongly recommended to practice recovery procedures on non-production resources and data, as part of standard business continuity operation preparation.

- Define IaC artifacts to re-establish configuration settings and capabilities of an Azure Cosmos DB backup restore.

- Evaluate and apply the Azure Security Baseline control guidance for Azure Cosmos DB Backup and Recovery.
  - BR-1: Ensure regular automated backups
  - BR-3: Validate all backups including customer-managed keys
  - BR-4, Mitigate risk of lost keys

- For analytical workloads requiring multi-region availability, use the Azure Cosmos DB Analytical Store, which applies a column format for optimized analytical queries.

Relational data technologies

For scenarios with a highly relational data model or dependencies on existing relational technologies, the use of Azure Cosmos DB in a multi-region write configuration might not be directly applicable. In such cases, it's vital that used relational technologies are designed and configured to uphold the multi-region active-active aspirations of an application design.

Azure provides many managed relational data platforms, including Azure SQL Database and Azure Database for common OSS relational solutions, including MySQL, PostgreSQL, and MariaDB. The design considerations and recommendations within this section will therefore focus on the optimal usage of Azure SQL Database and Azure Database OSS flavors to maximize reliability and global availability.

Design considerations

- Whilst relational data technologies can be configured to easily scale read operations, writes are typically constrained to go through a single primary instance, which places a significant constraint on scalability and performance.

- Sharding can be applied to distribute data and processing across multiple identical structured databases, partitioning databases horizontally to navigate platform constraints.
  - For example, sharding is often applied in multi-tenant SaaS platforms to isolate groups of tenants into distinct data platform constructs.

Azure SQL Database

- Azure SQL Database provides a fully managed database engine that is always running on the latest stable version of the SQL Server database engine and underlying Operating System.
  - Provides intelligent features such as performance tuning, threat monitoring, and vulnerability assessments.

- Azure SQL Database provides built-in regional high availability and turnkey geo-replication to distribute read-replicas across Azure regions.
  - With geo-replication, secondary database replicas remain read-only until a failover is initiated.

  - Up to four secondaries are supported in the same or different regions.
  - Secondary replicas can also be used for read-only query access to optimize read performance.
  - Failover must be initiated manually but can be wrapped in automated operational procedures.

- Azure SQL Database provides Auto Failover Groups, which replicates databases to a secondary server and allows for transparent failover if a failure.
  - Auto-failover groups support geo-replication of all databases in the group to only one secondary server or instance in a different region.
  - Auto-failover groups aren't currently supported in the Hyperscale service tier.
  - Secondary databases can be used to offload read traffic.

- Premium or Business Critical service tier database replicas can be distributed across Availability Zones at no extra cost.
  - The control ring is also duplicated across multiple zones as three gateway rings (GW).
    - The routing to a specific gateway ring is controlled by Azure Traffic Manager.
  - When using the Business Critical tier, zone redundant configuration is only available when the Gen5 compute hardware is selected.

- Azure SQL Database offers a baseline 99.99% availability SLA across all of its service tiers, but provides a higher 99.995% SLA for the Business Critical or Premium tiers in regions that support availability zones.
  - Azure SQL Database Business Critical or Premium tiers not configured for Zone Redundant Deployments have an availability SLA of 99.99%.

- When configured with geo-replication, the Azure SQL Database Business Critical tier provides a Recovery Time Objective (RTO) of 30 seconds for 100% of deployed hours.

- When configured with geo-replication, the Azure SQL Database Business Critical tier has a Recovery point Objective (RPO) of 5 seconds for 100% of deployed hours.

- Azure SQL Database Hyperscale tier, when configured with at least two replicas, has an availability SLA of 99.99%.

- Compute costs associated with Azure SQL Database can be reduced using a Reservation Discount.
  - It's not possible to apply reserved capacity for DTU-based databases.

- Point-in-time restore can be used to return a database and contained data to an earlier point in time.
- Geo-restore can be used to recover a database from a geo-redundant backup.

Azure Database For PostgreSQL

- Azure Database For PostgreSQL is offered in three different deployment options:
  - Single Server, SLA 99.99%
  - Flexible Server, which offers Availability Zone redundancy, SLA 99.99%
  - Hyperscale (Citus), SLA 99.95% when High Availability mode is enabled.
- Hyperscale (Citus) provides dynamic scalability through sharding without application changes.
  - Distributing table rows across multiple PostgreSQL servers is key to ensure scalable queries in Hyperscale (Citus).
  - Multiple nodes can collectively hold more data than a traditional database, and in many cases can use worker CPUs in parallel to optimize costs.
- Autoscale can be configured through runbook automation to ensure elasticity in response to changing traffic patterns.
- Flexible server provides cost efficiencies for non-production workloads through the ability to stop/start the server, and a burstable compute tier that is suitable for workloads that don't require continuous compute capacity.
- There's no additional charge for backup storage for up to 100% of total provisioned server storage.
  - Additional consumption of backup storage is charged according to consumed GB/month.
- Compute costs associated with Azure Database for PostgreSQL can be reduced using either a Single Server Reservation Discount or Hyperscale (Citus) Reservation Discount.

Design Recommendations

- Consider sharding to partition relational databases based on different application and data contexts, helping to navigate platform constraints, maximize scalability and availability, and fault isolation.
  - This recommendation is particularly prevalent when the application design considers three or more Azure regions since relational technology constraints can significantly hinder globally distributed data platforms.

  - Sharding isn't appropriate for all application scenarios, so a contextualized evaluation is required.

- Prioritize the use of Azure SQL Database where relational requirements exist due to its maturity on the Azure platform and wide array of reliability capabilities.

Azure SQL Database

- Use the Business-Critical service tier to maximize reliability and availability, including access to critical resiliency capabilities.

- Use the vCore based consumption model to facilitate the independent selection of compute and storage resources, tailored to workload volume and throughput requirements.
  - Ensure a defined capacity model is applied to inform compute and storage resource requirements.
    - Consider Reserved Capacity to provide potential cost optimizations.

- Configure the Zone-Redundant deployment model to spread Business Critical database replicas within the same region across Availability Zones.

- Use Active Geo-Replication to deploy readable replicas within all deployment regions (up to four).

- Use Auto Failover Groups to provide transparent failover to a secondary region, with geo-replication applied to provide replication to additional deployment regions for read optimization and database redundancy.
  - For application scenarios limited to only two deployment regions, the use of Auto Failover Groups should be prioritized.

- Consider automated operational triggers, based on alerting aligned to the application health model, to conduct failovers to geo-replicated instances if a failure impacting the primary and secondary within the Auto Failover Group.

> ⓘ **Important**
>
> For applications considering more than four deployment regions, serious consideration should be given to application scoped sharding or refactoring the application to support multi-region write technologies, such as Azure Cosmos DB. However, if this isn't feasible within the application workload scenario, it's advised to elevate a region within a single geography to a primary status encompassing a geo-replicated instance to more evenly distributed read access.

- Configure the application to query replica instances for read queries to optimize read performance.

- Use Azure Monitor and Azure SQL Analytics for near real-time operational insights in Azure SQL DB for the detection of reliability incidents.

- Use Azure Monitor to evaluate usage for all databases to determine if they have been sized appropriately.
  - Ensure CD pipelines consider load testing under representative load levels to validate appropriate data platform behavior.

- Calculate a health metric for database components to observe health relative to business requirements and resource utilization, using monitoring and alerts to drive automated operational action where appropriate.
  - Ensure key query performance metrics are incorporated so swift action can be taken when service degradation occurs.

- Optimize queries, tables, and databases using Query Performance Insights and common performance recommendations provided by Microsoft.

- Implement retry logic using the SDK to mitigate transient errors impacting Azure SQL Database connectivity.

- Prioritize the use of service-managed keys when applying server-side Transparent Data Encryption (TDE) for at-rest encryption.
  - If customer-managed keys or client-side (AlwaysEncrypted) encryption is required, ensure keys are appropriately resilient with backups and automated rotation facilities.

- Consider the use of point-in-time restore as an operational playbook to recover from severe configuration errors.

Azure Database For PostgreSQL

- Flexible Server is recommended to use it for business critical workloads due to its Availability Zone support.

- When using Hyperscale (Citus) for business critical workloads, enable High Availability mode to receive the 99.95% SLA guarantee.

- Use the Hyperscale (Citus) server configuration to maximize availability across multiple nodes.

- Define a capacity model for the application to inform compute and storage resource requirements within the data platform.

- Consider the Hyperscale (Citus) Reservation Discount to provide potential cost optimizations.

Caching for Hot Tier Data

An in-memory caching layer can be applied to enhance a data platform by significantly increasing read throughput and improving end-to-end client response times for hot tier data scenarios.

Azure provides several services with applicable capabilities for caching key data structures, with Azure Cache for Redis positioned to abstract and optimize data platform read access. This section will therefore focus on the optimal usage of Azure Cache for Redis in scenarios where additional read performance and data access durability is required.

Design Considerations

- A caching layer provides additional data access durability since even if an outage impacting the underlying data technologies, an application data snapshot can still be accessed through the caching layer.

- In certain workload scenarios, in-memory caching can be implemented within the application platform itself.

Azure Cache for Redis

- Redis cache is an open source NoSQL key-value in-memory storage system.

- The Enterprise and Enterprise Flash tiers can be deployed in an active-active configuration across Availability Zones within a region and different Azure regions through geo-replication.
  - When deployed across at least three Azure regions and three or more Availability Zones in each region, with active geo-replication enabled for all Cache instances, Azure Cache for Redis provides an SLA of 99.999% for connectivity to one regional cache endpoint.
  - When deployed across three Availability Zones within a single Azure region a 99.99% connectivity SLA is provided.

- The Enterprise Flash tier runs on a combination of RAM and flash non-volatile memory storage, and while this introduces a small performance penalty it also enables very large cache sizes, up to 13TB with clustering.

- With geo-replication, charges for data transfer between regions will also be applicable in addition to the direct costs associated with cache instances.

- The Scheduled Updates feature doesn't include Azure updates or updates applied to the underlying VM operating system.

- There will be an increase in CPU utilization during a scale-out operation while data is migrated to new instances.

Design Recommendations

- Consider an optimized caching layer for 'hot' data scenarios to increase read throughput and improve response times.

- Apply appropriate policies for cache expiration and housekeeping to avoid runaway data growth.
  - Consider expiring cache items when the backing data changes.

Azure Cache for Redis

- Use the Premium or Enterprise SKU to maximize reliability and performance.
  - For scenarios with extremely large data volumes, the Enterprise Flash tier should be considered.
  - For scenarios where only passive geo-replication is required, the Premium tier can also be considered.

- Deploy replica instances using geo-replication in an active configuration across all considered deployment regions.

- Ensure replica instances are deployed across Availability Zones within each considered Azure region.

- Use Azure Monitor to evaluate Azure Cache for Redis.
  - Calculate a health score for regional cache components to observe health relative to business requirements and resource utilization.
  - Observe and alert on key metrics such as high CPU, high memory usage, high server load, and evicted keys for insights when to scale the cache.

- Optimize connection resilience by implementing retry logic, timeouts, and using a singleton implementation of the Redis connection multiplexer.

- Configure scheduled updates to prescribe the days and times that Redis Server updates are applied to the cache.

Analytical Scenarios

It's increasingly common for mission-critical applications to consider analytical scenarios as a means to drive additional value from encompassed data flows. Application and operational (AIOps) analytical scenarios therefore form a crucial aspect of highly reliable data platform.

Analytical and transactional workloads require different data platform capabilities and optimizations for acceptable performance within their respective contexts.

Description	Analytical	Transactional
Use Case	Analyze very large volumes of data ("big data")	Process very large volumes of individual transactions
Optimized for	Read queries and aggregations over many records	Near real-time Create/Read/Update/Delete (CRUD) queries over few records
Key Characteristics	- Consolidate from data sources of record - Column-based storage - Distributed storage - Parallel processing - Denormalized - Low concurrency reads and writes - Optimize for storage volume with compression	- Data source of record for application - Row-based Storage - Contiguous storage - Symmetrical processing - Normalized - High concurrency reads and writes, index updates - Optimize for fast data access with in-memory storage

Azure Synapse provides an enterprise analytical platform that brings together relational and non-relational data with Spark technologies, using built-in integration with Azure services such as Azure Cosmos DB to facilitate big data analytics. The design considerations and recommendations within this section will therefore focus on optimal Azure Synapse and Azure Cosmos DB usage for analytical scenarios.

Design Considerations

- Traditionally, large-scale analytical scenarios are facilitated by extracting data into a separate data platform optimized for subsequent analytical queries.
  - Extract, Transform, and Load (ETL) pipelines are used to extract data will consume throughput and impact transactional workload performance.
  - Running ETL pipelines infrequently to reduce throughput and performance impacts will result in analytical data that is less up-to-date.

  - ETL pipeline development and maintenance overhead increases as data transformations become more complex.
    - For example, if source data is frequently changed or deleted, ETL pipelines must account for those changes in the target data for analytical queries through an additive/versioned approach, dump and reload, or in-place changes on the analytical data. Each of these approaches will have derivative impact, such as index re-creation or update.

Azure Cosmos DB

- Analytical queries run on Azure Cosmos DB transactional data will typically aggregate across partitions over large volumes of data, consuming significant Request Unit (RU) throughput, which can impact the performance of surrounding transactional workloads.

- The Azure Cosmos DB Analytical Store provides a schematized, fully isolated column-oriented data store that enables large-scale analytics on Azure Cosmos DB data from Azure Synapse without impact to Azure Cosmos DB transactional workloads.
  - When an Azure Cosmos DB Container is enabled as an Analytical Store, a new column store is internally created from the operational data in the Container. This column store is persisted separately from the row-oriented transaction store for the container.
  - Create, Update and Delete operations on the operational data are automatically synced to the analytical store, so no change feed or ETL processing is required.
  - Data sync from the operational to the analytical store doesn't consume throughput Request Units (RUs) provisioned on the Container or Database. There's no performance impact on transactional workloads. Analytical Store doesn't require allocation of additional RUs on an Azure Cosmos DB Database or Container.
  - Auto-Sync is the process where operational data changes are automatically synced to the Analytical Store. Auto-Sync latency is usually less than two (2) minutes.
    - Auto-Sync latency can be up to five (5) minutes for a Database with shared throughput and a large number of Containers.
    - As soon as Auto-Sync completes, the latest data can be queried from Azure Synapse.
  - Analytical Store storage uses a consumption-based pricing model that charges for volume of data and number of read and write operations. Analytical store pricing is separate from transactional store pricing.

- Using Azure Synapse Link, the Azure Cosmos DB Analytical Store can be queried directly from Azure Synapse. This enables no-ETL, Hybrid Transactional-Analytical Processing (HTAP) from Synapse, so that Azure Cosmos DB data can be queried together with other analytical workloads from Synapse in near real-time.

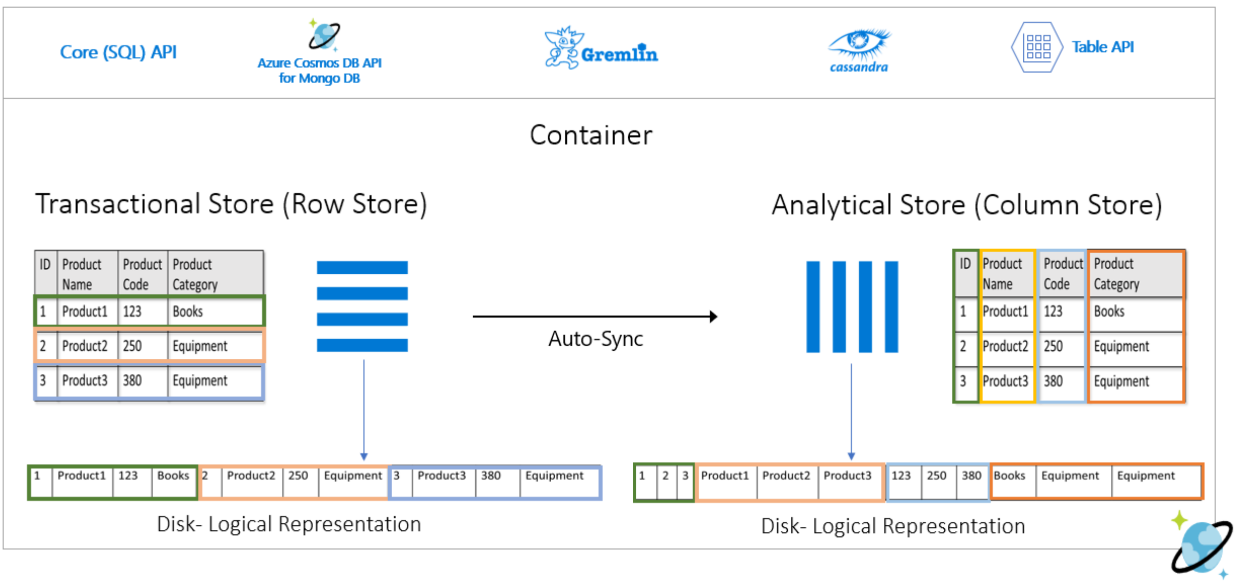
- The Azure Cosmos DB Analytical Store isn't partitioned by default.
  - For certain query scenarios, performance will improve by partitioning Analytical Store data using keys that are frequently used in query predicates.
  - Partitioning is triggered by a job in Azure Synapse that runs a Spark notebook using Synapse Link, which loads the data from the Azure Cosmos DB Analytical Store and writes it into the Synapse partitioned store in the primary storage account of the Synapse workspace.

- Azure Synapse Analytics SQL Serverless pools can query the Analytical Store through automatically updated views or via `SELECT / OPENROWSET` commands.

- Azure Synapse Analytics Spark pools can query the Analytical Store through automatically updated Spark tables or the `spark.read` command.

- Data can also be copied from the Azure Cosmos DB Analytical Store into a dedicated Synapse SQL pool using Spark, so that provisioned Azure Synapse SQL pool resources can be used.

- Azure Cosmos DB Analytical Store data can be queried with Azure Synapse Spark.
  - Spark notebooks allow for Spark dataframe combinations to aggregate and transform Azure Cosmos DB analytical data with other data sets, and use other advanced Synapse Spark functionality including writing transformed data to other stores or training AIOps Machine Learning models.

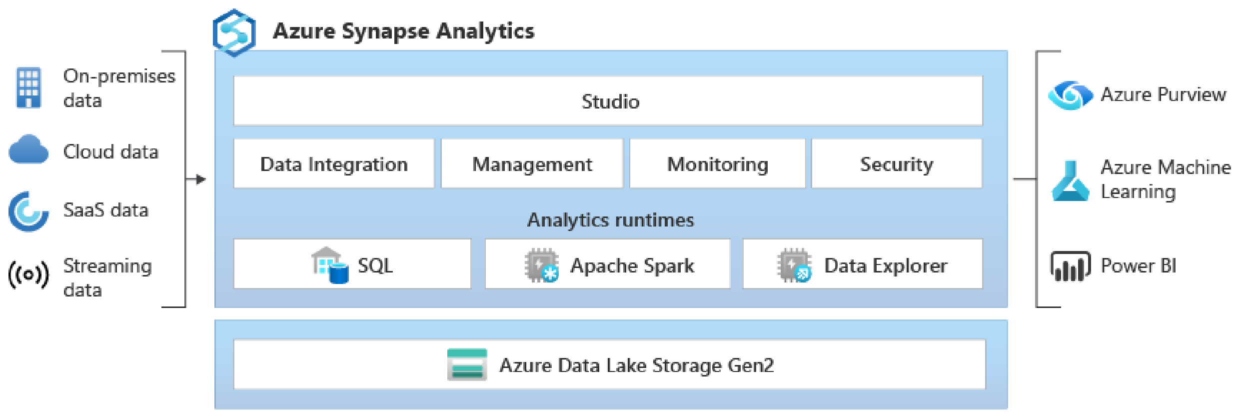


- The Azure Cosmos DB change feed can also be used to maintain a separate secondary data store for analytical scenarios.

Azure Synapse

- Azure Synapse brings together analytics capabilities including SQL data warehousing, Spark big data, and Data Explorer for log and time series analytics.
  - Azure Synapse uses *linked services* to define connections to other services, such as Azure Storage.
  - Data can be ingested into Synapse Analytics via Copy activity from supported sources. This permits data analytics in Synapse without impacting the source data store, but adds time, cost and latency overhead due to data transfer.
  - Data can also be queried in-place in supported external stores, avoiding the overhead of data ingestion and movement. Azure Storage with Data Lake Gen2 is a supported store for Synapse and Log Analytics exported data can be queried via Synapse Spark.

- Azure Synapse Studio unites ingestion and querying tasks.
  - Source data, including Azure Cosmos DB Analytical Store data and Log Analytics Export data, are queried and processed in order to support business intelligence and other aggregated analytical use cases.



Design Recommendations

- Ensure analytical workloads don't impact transactional application workloads to maintain transactional performance.

Application Analytics

- Use Azure Synapse Link with Azure Cosmos DB Analytical Store to perform analytics on Azure Cosmos DB operational data by creating an optimized data store, which won't impact transactional performance.
  - Enable Azure Synapse Link on Azure Cosmos DB accounts.

- Create a container enabled for Analytical Store, or enable an existing Container for Analytical Store.
- Connect the Azure Synapse workspace to the Azure Cosmos DB Analytical Store to enable analytical workloads in Azure Synapse to query Azure Cosmos DB data. Use a connection string with a read-only Azure Cosmos DB key.

- Prioritize Azure Cosmos DB Analytical Store with Azure Synapse Link instead of using the Azure Cosmos DB change feed to maintain an analytical data store.
  - The Azure Cosmos DB change feed may be suitable for very simple analytical scenarios.

AIOps and Operational Analytics

- Create a single Azure Synapse workspace with linked services and data sets for each source Azure Storage account where operational data from resources are sent to.

- Create a dedicated Azure Storage account and use it as the workspace primary storage account to store the Synapse workspace catalog data and metadata. Configure it with hierarchical namespace to enable Azure Data Lake Gen2.
  - Maintain separation between the source analytical data and Synapse workspace data and metadata.
    - Do not use one of the regional or global Azure Storage accounts where operational data is sent.

Next step

Review the considerations for networking considerations.

Network and connectivity

Networking and connectivity for mission-critical workloads on Azure

Article • 02/01/2023

Networking is a fundamental area for a mission-critical application, given the recommended globally distributed active-active design approach.

This design area explores various network topology topics at an application level, considering requisite connectivity and redundant traffic management. More specifically, it highlights critical considerations and recommendations intended to inform the design of a secure and scalable global network topology for a mission-critical application.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**

Global traffic routing

The use of multiple active regional deployment stamps requires a global routing service to distribute traffic to each active stamp.

Azure Front Door, Azure Traffic Manager, and Azure Standard Load Balancer provide the needed routing capabilities to manage global traffic across a multi-region application.

Alternatively, third-party globally routing technologies can be considered. These options can almost seamlessly be swapped in to replace or extend the use of Azure-native global routing services. Popular choices include routing technologies by CDN providers.

This section explores the key differences Azure routing services to define how each can be used to optimize different scenarios.

Design considerations

- A routing service bound to a single region represents a single-point-of-failure and a significant risk with regard to regional outages.

- If the application workload encompasses client control, such as with mobile or desktop client applications, it's possible to provide service redundancy within client routing logic.
  - Multiple global routing technologies, such as Azure Front Door and Azure Traffic Manager, can be considered in parallel for redundancy, with clients configured to fail over to an alternative technology when certain failure conditions are met. The introduction of multiple global routing services introduces significant complexities around edge caching and Web Application Firewall capabilities, and certificate management for SSL offload and application validation for ingress paths.
  - Third-party technologies can also be considered, providing global routing resiliency to all levels of Azure platform failures.

- Capability disparity between Azure Front Door and Traffic Manager means that if the two technologies are positioned alongside one another for redundancy, a different ingress path or design changes would be required to ensure a consistent and acceptable level of service is maintained.

- Azure Front Door and Azure Traffic Manager are globally distributed services with built-in multi-region redundancy and availability.
  - Hypothetical failure scenarios of a scale large enough to threaten the global availability of these resilient routing services presents a broader risk to the application in terms of cascading and correlated failures.
    - Failure scenarios of this scale are only feasibly caused by shared foundational services, such as Azure DNS or Azure AD, which serve as global platform dependencies for almost all Azure services. If a redundant Azure technology is applied, it's likely that the secondary service will also be experiencing unavailability or a degraded service.
    - Global routing service failure scenarios are highly likely to significantly impact many other services used for key application components through interservice dependencies. Even if a third-party technology is used, the application will likely be in an unhealthy state due to the broader impact of the underlying issue, meaning that routing to application endpoints on Azure will provide little value anyway.

- Global routing service redundancy provides mitigation for an extremely small number of hypothetical failure scenarios, where the impact of a global outage is constrained to the routing service itself.

  To provide broader redundancy to global outage scenarios, a multi-cloud active-active deployment approach can be considered. A multi-cloud active-active deployment approach introduces significant operational complexities, which pose significant resiliency risks, likely far outweighing the hypothetical risks of a global outage.

- For scenarios where client control isn't possible, a dependency must be taken on a single global routing service to provide a unified entry point for all active deployment regions.
  - When used in isolation they represent a single-point-of-failure at a service level due to global dependencies, even though built-in multi-region redundancy and availability are provided.
  - The SLA provided by the selected global routing service represents the maximum attainable composite SLA, regardless of how many deployment regions are considered.

- When client control isn't possible, operational mitigations can be considered to define a process for migrating to a secondary global routing service if a global outage disables the primary service. Migrating from one global routing service to another is typically a lengthy process lasting several hours, particularly where DNS propagation is considered.

- Some third-party global routing services provide a 100% SLA. However, the historic and attainable SLA provided by these services is typically lower than 100%.
  - While these services provide financial reparations for unavailability, it comes of little significance when the impact of unavailability is significant, such as with safety-critical scenarios where human life is ultimately at stake. Technology redundancy or sufficient operational mitigations should therefore still be considered even when the advertised legal SLA is 100%.

Azure Front Door

- Azure Front Door provides global HTTP/S load balancing and optimized connectivity using the Anycast protocol with split TCP to take advantage of the Microsoft global backbone network.
  - A number of connections are maintained for each of the backend endpoints.
  - Incoming client requests are first terminated at the edge node closest to the originating client.
  - After any required traffic inspection, requests are either forwarded over the Microsoft backbone to the appropriate backend using existing connections, or served from the internal cache of an edge node.
  - This approach is very efficient in spreading high traffic volumes over the backend connections.

- Provides a built-in cache that serves static content from edge nodes. In many use cases, this can also eliminate the need for a dedicated Content Delivery Network

(CDN).

- Azure Web Application Firewall (WAF) can be used on Azure Front Door, and since it's deployed to Azure network edge locations around the globe, every incoming request delivered by Front Door in inspected at the network edge.

- Azure Front Door protects application endpoints against DDoS attacks using Azure DDoS protection Basic. Azure DDoS Standard provides additional and more advanced protection and detection capabilities and can be added as an additional layer to Azure Front Door.

- Azure Front Door offers a fully managed certificate service. Enables TLS connection security for endpoints without having to manage the certificate lifecycle.

- Azure Front Door Premium supports private endpoints, enabling traffic to flow from the internet directly onto Azure virtual networks. This would eliminate the need of using public IPs on the VNet for making the backends accessible via Azure Front Door Premium.

- Azure Front Door relies on health probes and backend health endpoints (URLs) that are called on an interval basis to return an HTTP status code reflecting if the backend is operating normally, with an HTTP 200 (OK) response reflecting a healthy status. As soon as a backend reflects an unhealthy status, from the perspective of a certain edge node, that edge node will stop sending requests there. Unhealthy backends are therefore transparently removed from traffic circulation without any delay.

- Supports HTTP/S protocols only.

- The Azure Front Door WAF and Application Gateway WAF provide a slightly different feature set, though both support built-in and custom rules and can be set to operate in either detection mode or prevention mode.

- The Front Door backend IP space may change, but Microsoft will ensure integration with Azure IP Ranges and Service Tags. It's possible to subscribe to Azure IP Ranges and Service Tags to receive notifications about any changes or updates.

- Azure Front Door supports various load distribution configurations:
  - Latency-based: the default setting that routes traffic to the "closest" backend from the client; based on request latency.
  - Priority-based: useful for active-passive setups, where traffic must always be sent to a primary backend unless it's not available.

  - Weighted: applicable for canary deployments in which a certain percentage of traffic is sent to a specific backend. If multiple backends have the same weights assigned, latency-based routing is used.

- By default Azure Front Door uses latency-based routing that can lead to situations where some backends get a lot more incoming traffic than others, depending on where clients originate from.

- If a series of client requests must be handled by the same backend, Session Affinity can be configured on the frontend. It uses a client-side cookie to send subsequent requests to the same backend as the first request, provided the backend is still available.

Azure Traffic Manager

- Azure Traffic Manager is a DNS redirection service.
  - The actual request payload isn't processed, but instead Traffic Manager returns the DNS name of one of the backends it the pool, based on configured rules for the selected traffic routing method.
  - The backend DNS name is then resolved to its final IP address that is subsequently directly called by the client.

- The DNS response is cached and reused by the client for a specified Time-To-Live (TTL) period, and requests made during this period will go directly to the backend endpoint without Traffic Manager interaction. Eliminates the extra connectivity step that provides cost benefits compared to Front Door.

- Since the request is made directly from the client to the backend service, any protocol supported by the backend can be leveraged.

- Similar to Azure Front Door, Azure Traffic Manager also relies on health probes to understand if a backend is healthy and operating normally. If another value is returned or nothing is returned, the routing service recognizes ongoing issues and will stop routing requests to that specific backend.
  - However, unlike with Azure Front Door this removal of unhealthy backends isn't instantaneous since clients will continue to create connections to the unhealthy backend until the DNS TTL expires and a new backend endpoint is requested from the Traffic Manager service.
  - In addition, even when the TTL expires, there no guarantee that public DNS servers will honor this value, so DNS propagation can actually take much longer to occur. This means that traffic may continue to be sent to the unhealthy endpoint for a sustained period of time.

Azure Standard Load Balancer

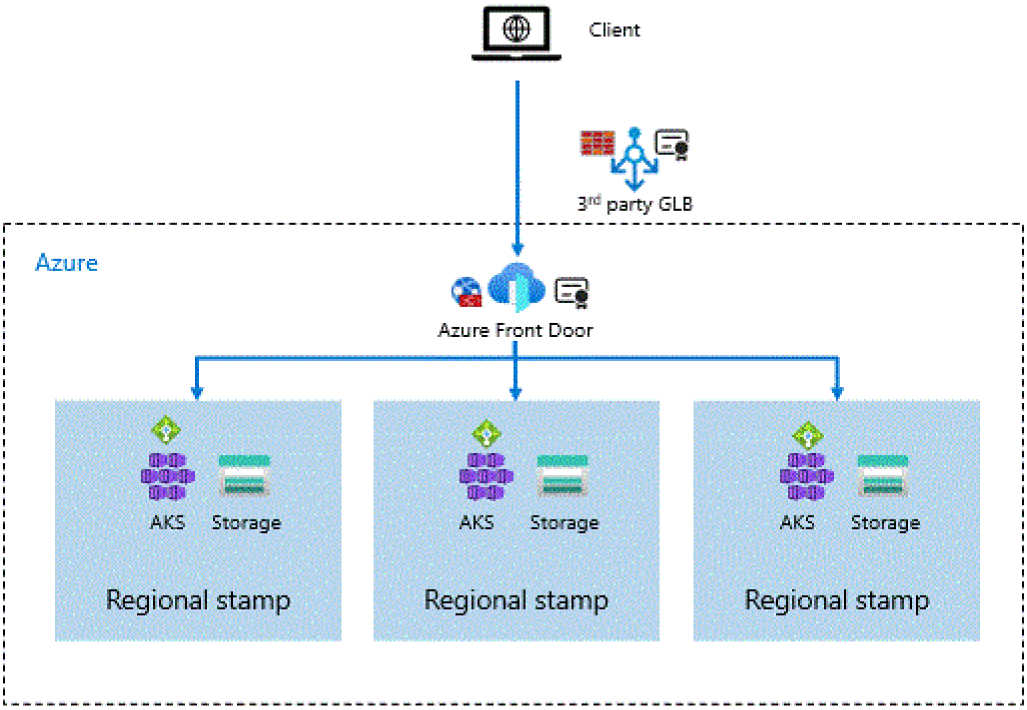
> ⓘ **Important**
>
> Cross-Region Standard Load Balancer is available in preview with **technical limitations**. This option isn't recommended for mission-critical workloads.

Design recommendations

- Use Azure Front Door as the primary global traffic routing service for HTTP/S scenarios. Azure Front Door is strongly advocated for HTTP/S workloads as it provides optimized traffic routing, transparent fail over, private backend endpoints (with the Premium SKU), edge caching and integration with Web Application Firewall (WAF).

- For application scenarios where client control is possible, apply client side routing logic to consider failover scenarios where the primary global routing technology fails. Two or more global routing technologies should be positioned in parallel for added redundancy, if single service SLA isn't sufficient. Client logic is required to route to the redundant technology in case of a global service failure.
  - Two distinct URLs should be used, with one applied to each of the different global routing services to simplify the overall certificate management experience and routing logic for a failover.
  - Prioritize the use of third-party routing technologies as the secondary failover service, since this will mitigate the largest number of global failure scenarios and the capabilities offered by industry leading CDN providers will allow for a consistent design approach.
  - Consideration should also be given to directly routing to a single regional stamp rather than a separate routing service. While this will result in a degraded level of service, it represents a far simpler design approach.

This image shows a redundant global load balancer configuration with client failover using Azure Front Door as primary global load balancer.



ⓘ **Important**

To truly mitigate the risk of global failures within the Azure platform, a multi-cloud active-active deployment approach should be considered, with active deployment stamps hosted across two or more cloud providers and redundant third-party routing technologies used for global routing.

Azure can effectively be integrated with other cloud platforms. However, it's strongly recommended not to apply a multi-cloud approach because it introduces significant operational complexity, with different deployment stamp definitions and representations of operational health across the different cloud platforms. This complexity in-turn introduces numerous resiliency risks within the normal operation of the application, which far outweigh the hypothetical risks of a global platform outage.

- Although not recommended, for HTTP(s) workloads using Azure Traffic Manager for global routing redundancy to Azure Front Door, consider offloading Web Application Firewall (WAF) to Application Gateway for acceptable traffic flowing through Azure Front Door.
  - This will introduce an additional failure point to the standard ingress path, an additional critical-path component to manage and scale, and will also incur additional costs to ensure global high-availability. It will, however, greatly simplify the failure scenario by providing consistency between the acceptable and not acceptable ingress paths through Azure Front Door and Azure Traffic

Manager, both in terms of WAF execution but also private application endpoints.
    - The loss of edge caching in a failure scenario will impact overall performance, and this must be aligned with an acceptable level of service or mitigating design approach. To ensure a consistent level of service, consider offloading edge caching to a third-party CDN provider for both paths.

It's recommended to consider a third-party global routing service in place of two Azure global routing services. This provides the maximum level of fault mitigation and a more simple design approach since most industry leading CDN providers offer edge capabilities largely consistent with that offered by Azure Front Door.

Azure Front Door

- Use the Azure Front Door managed certificate service to enable TLS connections, and remove the need to manage certificate lifecycles.

- Use the Azure Front Door Web Application Firewall (WAF) to provide protection at the edge from common web exploits and vulnerabilities, such as SQL injection.

- Use the Azure Front Door built-in cache to serve static content from edge nodes.
    - In most cases this will also eliminate the need for a dedicated Content Delivery Network (CDN).

- Configure the application platform ingress points to validate incoming requests through header based filtering using the *X-Azure-FDID* to ensure all traffic is flowing through the configured Front Door instance. Consider also configuring IP ACLing using Front Door Service Tags to validate traffic originates from the Azure Front Door backend IP address space and Azure infrastructure services. This will ensure traffic flows through Azure Front Door at a service level, but header based filtering will still be required to ensure the use of a configured Front Door instance.

- Define a custom TCP health endpoint to validate critical downstream dependencies within a regional deployment stamp, including data platform replicas, such as Azure Cosmos DB in the example provided by the foundational reference implementation. If one or more dependencies becomes unhealthy, the health probe should reflect this in the response returned so that the entire regional stamp can be taken out of circulation.

- Ensure health probe responses are logged and ingest all operational data exposed by Azure Front Door into the global Log Analytics workspace to facilitate a unified data sink and single operational view across the entire application.

- Unless the workload is extremely latency sensitive, spread traffic evenly across all considered regional stamps to most effectively use deployed resources.
  - To achieve this, set the "Latency Sensitivity (Additional Latency)" parameter to a value that is high enough to cater for latency differences between the different regions of the backends. Ensure a tolerance that is acceptable to the application workload regarding overall client request latency.

- Don't enable Session Affinity unless it's required by the application, since it can have a negative impact the balance of traffic distribution. With a fully stateless application, if the recommended mission-critical application design approach is followed, any request could be handled by any of the regional deployments.

Azure Traffic Manager

- Use Traffic Manager for non HTTP/S scenarios as a replacement to Azure Front Door. Capability differences will drive different design decisions for cache and WAF capabilities, and TLS certificate management.

- WAF capabilities should be considered within each region for the Traffic Manager ingress path, using Azure Application Gateway.

- Configure a suitably low TTL value to optimize the time required to remove an unhealthy backend endpoint from circulation in the event that backend becomes unhealthy.

- Similar to with Azure Front Door, a custom TCP health endpoint should be defined to validate critical downstream dependencies within a regional deployment stamp, which should be reflected in the response provided by health endpoints.

  However, for Traffic Manager additional consideration should be given to service level regional fail over. such as 'dog legging', to mitigate the potential delay associated with the removal of an unhealthy backend due to dependency failures, particularly if it's not possible to set a low TTL for DNS records.

- Consideration should be given to third-party CDN providers in order to achieve edge caching when using Azure Traffic Manager as a primary global routing service. Where edge WAF capabilities are also offered by the third-party service, consideration should be given to simplify the ingress path and potentially remove the need for Application Gateway.

Application delivery services

The network ingress path for a mission-critical application must also consider application delivery services to ensure secure, reliable, and scalable ingress traffic.

This section builds on global routing recommendations by exploring key application delivery capabilities, considering relevant services such as Azure Standard Load Balancer, Azure Application Gateway, and Azure API Management.

Design considerations

- TLS encryption is critical to ensure the integrity of inbound user traffic to a mission-critical application, with **TLS Offloading** applied only at the point of a stamp's ingress to decrypt incoming traffic. TLS Offloading Requires the private key of the TLS certificate to decrypt traffic.

- A **Web Application Firewall** provides protection against common web exploits and vulnerabilities, such as SQL injection or cross site scripting, and is essential to achieve the maximum reliability aspirations of a mission-critical application.

- Azure WAF provides out-of-the-box protection against the top 10 OWASP vulnerabilities using managed rule sets.
  - Custom rules can also be added to extend the managed rule set.
  - Azure WAF can be enabled within either Azure Front Door, Azure Application Gateway, or Azure CDN (currently in public preview).
    - The features offered on each of the services differ slightly. For example, the Azure Front Door WAF provides rate limiting, geo-filtering and bot protection, which aren't yet offered within the Application Gateway WAF. However, they all support both built-in and custom rules and can be set to operate in detection mode or prevention mode.
    - The roadmap for Azure WAF will ensure a consistent WAF feature set is provided across all service integrations.

- Third-party WAF technologies such as NVAs and advanced ingress controllers within Kubernetes can also be considered to provide requisite vulnerability protection.

- Optimal WAF configuration typically requires fine tuning, regardless of the technology used.

  Azure Front Door

- Azure Front Door only accepts HTTP and HTTPS traffic, and will only process requests with a known `Host` header. This protocol blocking helps to mitigate volumetric attacks spread across protocols and ports, and DNS amplification and TCP poisoning attacks.

- Azure Front Door is a global Azure resource so configuration is deployed globally to all edge locations.
  - Resource configuration can be distributed at a massive scale to handle hundreds of thousands of requests per second.
  - Updates to configuration, including routes and backend pools, are seamless and won't cause any downtime during deployment.

- Azure Front Door provides both a fully managed certificate service and a bring-your-own-certificate method for the client-facing SSL certificates. The fully managed certificate service provides a simplified operational approach and helps to reduce complexity in the overall design by performing certificate management within a single area of the solution.

- Azure Front Door auto-rotates "Managed" certificates at least 60 days ahead of certificate expiration to protect against expired certificate risks. If self-managed certificates are used, updated certificates should be deployed no later than 24 hours prior to expiration of the existing certificate, otherwise clients may receive expired certificate errors.

- Certificate updates will only result in downtime if Azure Front Door is switched between "Managed" and "Use Your Own Certificate".

- Azure Front Door is protected by Azure DDoS Protection Basic, which is integrated into Front Door by default. This provides always-on traffic monitoring, real-time mitigation, and also defends against common Layer 7 DNS query floods or Layer 3/4 volumetric attacks.
  - These protections help to maintain Azure Front Door availability even when faced with a DDoS attack. Distributed Denial of Service (DDoS) attacks can render a targeted resource unavailable by overwhelming it with illegitimate traffic.

- Azure Front Door also provides WAF capabilities at a global traffic level, while Application Gateway WAF must be provided within each regional deployment stamp. Capabilities include firewall rulesets to protect against common attacks, geo-filtering, address blocking, rate limiting, and signature matching.

Azure Load Balancer

- The Azure Basic Load Balancer SKU isn't backed by an SLA and has several capability constraints compared to the Standard SKU.

Design recommendations

- Perform TLS Offloading in as few places as possible in order to maintain security whilst simplifying the certificate management lifecycle.

- Use encrypted connections (e.g. HTTPS) from the point where TLS offloading occurs to the actual application backends. Application endpoints won't be visible to end users, so Azure-managed domains, such as `azurewebsites.net` or `cloudapp.net`, can be used with managed certificates.

- For HTTP(S) traffic, ensure WAF capabilities are applied within the ingress path for all publicly exposed endpoints.

- Enable WAF capabilities at a single service location, either globally with Azure Front Door or regionally with Azure Application Gateway, since this simplifies configuration fine tuning and optimizes performance and cost.

  Configure WAF in Prevention mode to directly block attacks. Only use WAF in Detection mode (i.e. only logging but not blocking suspicious requests) when the performance penalty of Prevention mode is too high. The implied additional risk must be fully understood and aligned to the specific requirements of the workload scenario.

- Prioritize the use of Azure Front Door WAF since it provides the richest Azure-native feature set and applies protections at the global edge, which simplifies the overall design and drives further efficiencies.

- Use Azure API Management only when exposing a large number of APIs to external clients or different application teams.

- Use the Azure Standard Load Balancer SKU for any internal traffic distribution scenario within micros-service workloads.
  - Provides an SLA of 99.99% when deployed across Availability Zones.
  - Provides critical capabilities such as diagnostics or outbound rules.

- Use Azure DDoS Network Protection to help protect public endpoints hosted within each application virtual network.

Caching and static content delivery

Special treatment of static content like images, JavaScript, CSS and other files can have a significant impact on the overall user experience as well as on the overall cost of the solution. Caching static content at the edge can speed up the client load times which results in a better user experience and can also reduce the cost for traffic, read operations and computing power on backend services involved.

Design considerations

- Not all content that a solution makes available over the Internet is generated dynamically. Applications serve both static assets (images, JavaScript, CSS, localization files, etc.) and dynamic content.
- Workloads with frequently accessed static content benefit greatly from caching since it reduces the load on backend services and reduces content access latency for end users.
- Caching can be implemented natively within Azure using either Azure Front Door or Azure Content Delivery Network (CDN).
  - Azure Front Door provides Azure-native edge caching capabilities and routing features to divide static and dynamic content.
    - By creating the appropriate routing rules in Azure Front Door, `/static/*` traffic can be transparently redirected to static content.
  - More complex caching scenarios can be implemented using the Azure CDN service to establish a full-fledged content delivery network for significant static content volumes.
    - The Azure CDN service will likely be more cost effective, but does not provide the same advanced routing and Web Application Firewall (WAF) capabilities which are recommended for other areas of an application design. It does, however, offer further flexibility to integrate with similar services from third-party solutions, such as Akamai and Verizon.
  - When comparing the Azure Front Door and Azure CDN services, the following decision factors should be explored:
    - Can required caching rules be accomplished using the rules engine.
    - Size of the stored content and the associated cost.
    - Price per month for the execution of the rules engine (charged per request on Azure Front Door).
    - Outbound traffic requirements (price differs by destination).

Design recommendations

- Generated, static content like sized copies of image files that never or only rarely change can benefit from caching as well. Caching can be configured based on URL parameters and with varying caching duration.
- Separate the delivery of static and dynamic content to users and deliver relevant content from a cache to reduce load on backend services optimize performance

for end-users.
- Given the strong recommendation (Network and connectivity design area) to use Azure Front Door for global routing and Web Application Firewall (WAF) purposes, it's recommended to prioritize the use of Azure Front Door caching capabilities unless gaps exist.

Virtual network integration

A mission-critical application will typically encompass requirements for integration with other applications or dependent systems, which could be hosted on Azure, another public cloud, or on-premises data centers. This application integration can be accomplished using public-facing endpoints and the internet, or private networks through network-level integration. Ultimately, the method by which application integration is achieved will have a significant impact on the security, performance, and reliability of the solution, and strongly impacting design decisions within other design areas.

A mission-critical application can be deployed within one of three overarching network configurations, which determines how application integration can occur at a network level.

1. **Public** application **without** corporate network connectivity.
2. **Public** application **with** corporate network connectivity.
3. **Private** application **with** corporate network connectivity.

> ⊗ **Caution**
>
> When deploying within an Azure landing zone, configuration 1. should be deployed within an Online Landing Zone, while both 2) and 3) should be deployed within a Corp. Connected Landing Zone to facilitate network-level integration.

This section explores these network integration scenarios, layering in the appropriate use of Azure Virtual Networks and surrounding Azure networking services to ensure integration requirements are optimally satisfied.

Design Considerations

No virtual networks

- The simplest design approach is to not deploy the application within a virtual network.
  - Connectivity between all considered Azure services will be provided entirely through public endpoints and the Microsoft Azure backbone. Connectivity between public endpoints hosted on Azure will only traverse the Microsoft backbone and won't go over the public internet.
  - Connectivity to any external systems outside Azure will be provided by the public internet.

- This design approach adopts "identity as a security perimeter" to provide access control between the various service components and dependent solution. This may be an acceptable solution for scenarios that are less sensitive to security. All application services and dependencies are accessible through a public endpoint leaves them vulnerable to additional attack vectors orientated around gaining unauthorized access.

- This design approach is also not applicable for all Azure services, since many services, such as AKS, have a hard requirement for an underlying virtual network.

Isolated virtual networks

- To mitigate the risks associated with unnecessary public endpoints, the application can be deployed within a standalone network that isn't connected to other networks.

- Incoming client requests will still require a public endpoint to be exposed to the internet, however, all subsequent communication can be within the virtual network using private endpoints. When using Azure Front Door Premium, it's possible to route directly from edge nodes to private application endpoints.

- While private connectivity between application components will occur over virtual networks, all connectivity with external dependencies will still rely on public endpoints.
  - Connectivity to Azure platform services can be established with Private Endpoints if supported. If other external dependencies exist on Azure, such as another downstream application, connectivity will be provided through public endpoints and the Microsoft Azure backbone.
  - Connectivity to any external systems outside Azure would be provided by the public internet.

- For scenarios where there are no network integration requirements for external dependencies, deploying the solution within an isolated network environment provides maximum design flexibility. No addressing and routing constraints associated with broader network integration.

- Azure Bastion is a fully platform-managed PaaS service that can be deployed on a virtual network and provides secure RDP/SSH connectivity to Azure VMs. When you connect via Azure Bastion, virtual machines don't need a public IP address.

- The use of application virtual networks introduces significant deployment complexities within CI/CD pipelines, since both data plane and control plane access to resources hosted on private networks is required to facilitate application deployments.
  - Secure private network path must be established to allow CI/CD tooling to perform requisite actions.
  - Private build agents can be deployed within application virtual networks to proxy access to resources secured by the virtual network.

Connected virtual networks

- For scenarios with external network integration requirements, application virtual networks can be connected to other virtual networks within Azure, another cloud provider, or on-premises networks using a variety of connectivity options. For example, some application scenarios might consider application-level integration with other line-of-business applications hosted privately within an on-premises corporate network.

- The application network design must align with the broader network architecture, particularly concerning topics such as addressing and routing.

- Overlapping IP address spaces across Azure regions or on-premises networks will create major contention when network integration is considered.
  - A virtual network resource can be updated to consider additional address space, however, when a virtual network address space of a peered network changes a sync on the peering link is required, which will temporarily disable peering.
  - Azure reserves five IP addresses within each subnet, which should be considered when determining appropriate sizes for application virtual networks and encompassed subnets.
  - Some Azure services require dedicated subnets, such as Azure Bastion, Azure Firewall, or Azure Virtual Network Gateway. The size of these service subnets is very important, since they should be large enough to support all current instances of the service considering future scale requirements, but not so large as to unnecessarily waste addresses.

- When on-premises or cross-cloud network integration is required, Azure offers two different solutions to establish a secure connection.
  - An ExpressRoute circuit can be sized to provide bandwidths up to 100 Gbps.
  - A Virtual Private Network (VPN) can be sized to provide aggregated bandwidth up to 10 Gbps in hub and spoke networks, and up to 20 Gbps in Azure Virtual WAN.

> **Note**
>
> When deploying within an Azure landing zone, be aware that any required connectivity to on-premises networks should be provided by the landing zone implementation. The design can use ExpressRoute and other virtual networks in Azure using either Virtual WAN or a hub-and-spoke network design.

- The inclusion of additional network paths and resources introduces additional reliability and operational considerations for the application to ensure health is maintained.

Design recommendations

- It's recommended that mission-critical solutions are deployed within Azure virtual networks where possible to remove unnecessary public endpoints, limiting the application attack surface to maximize security and reliability.
  - Use Private Endpoints for connectivity to Azure platform services. Service Endpoints can be considered for services that don support Private Link, provided data exfiltration risks are acceptable or mitigated through alternative controls.

- For application scenarios that don't require corporate network connectivity, treat all virtual networks as ephemeral resources that are replaced when a new regional deployment is conducted.

- When connecting to other Azure or on-premises networks, application virtual networks shouldn't be treated as ephemeral since it creates significant complications where virtual network peering and virtual network gateway resources are concerned. All relevant application resources within the virtual network should continue to be ephemeral, with parallel subnets used to facilitate blue-green deployments of updated regional deployment stamps.

- In scenarios where corporate network connectivity is required to facilitate application integration over private networks, ensure that the IPv4 address space

used for regional application virtual networks doesn't overlap with other connected networks and is properly sized to facilitate required scale without needing to update the virtual network resource and incur downtime.
  - It's strongly recommended to only use IP addresses from the address allocation for private internet (RFC 1918).
    - For environments with a limited availability of private IP addresses (RFC 1918), consider using IPv6.
    - If the use of public IP address is required, ensure that only owned address blocks are used.
  - Align with organization plans for IP addressing in Azure to ensure that application network IP address space doesn't overlap with other networks across on-premises locations or Azure regions.
  - Don't create unnecessarily large application virtual networks to ensure that IP address space isn't wasted.

- Prioritize the use Azure CNI for AKS network integration, since it supports a richer feature set.

  - Consider Kubenet for scenarios with a limited rage available IP addresses to fit the application within a constrained address space.

  - Prioritize the use of the Azure CNI network plugin for AKS network integration and consider Kubenet for scenarios with a limited range of available IP addresses. See Micro-segmentation and kubernetes network policies for more details.

- For scenarios requiring on-premises network integration, prioritize the use Express Route to ensure secure and scalable connectivity.
  - Ensure the reliability level applied to the Express Route or VPN fully satisfies application requirements.
  - Multiple network paths should be considered to provide additional redundancy when required, such as cross connected ExpressRoute circuits or the use of VPN as a failover connectivity mechanism.

- Ensure all components on critical network paths are in line with the reliability and availability requirements of associated user flows, regardless of whether the management of these paths and associated component is delivered by the application team of central IT teams.

> ⓘ **Note**

When deploying within an Azure landing zone and integrating with a broader organizational network topology, consider the **network guidance** to ensure the foundational network is aligned with Microsoft best-practices.

- Use Azure Bastion or proxied private connections to access the data plane of Azure resources or perform management operations.

Internet egress

Internet egress is a foundational network requirement for a mission-critical application to facilitate external communication in the context of:

1. Direct application user interaction.
2. Application integration with external dependencies outside Azure.
3. Access to external dependencies required by the Azure services used by the application.

This section explores how internet egress can be achieved while ensuring security, reliability, and sustainable performance are maintained, highlighting key egress requirements for services recommended in a mission-critical context.

Design Considerations

- Many Azure services require access to public endpoints for various management and control plane functions to operate as intended.

- Azure provides different direct internet outbound connectivity methods, such as Azure NAT gateway or Azure Load Balancer, for virtual machines or compute instances on a virtual network.

- When traffic from inside a virtual network travels out to the Internet, Network Address Translation (NAT) must take place. This is a compute operation that occurs within the networking stack and that can therefore impact system performance.

- When NAT takes place at a small scale the performance impact should be negligible, however, if there are a large number of outbound requests network issues may occur. These issues typically come in the form of 'Source NAT (or SNAT) port exhaustion'.

- In a multi-tenant environment, such as Azure App Service, there's a limited number of outbound ports available to each instance. If these ports run out, no new outbound connections can be initiated. This issue can be mitigated by reducing the number of private/public edge traversals or by using a more scalable NAT solution such as the Azure NAT Gateway.

- In addition to NAT limitations, outbound traffic may also be subject to requisite security inspections.
  - Azure Firewall provides appropriate security capabilities to secure network egress.
  - Azure Firewall (or an equivalent NVA) can be used to secure Kubernetes egress requirements by providing granular control over outbound traffic flows.
- Large volumes of internet egress will incur data transfer charges.

Azure NAT Gateway

- Azure NAT Gateway supports 64,000 connections for TCP and UDP per assigned outbound IP address.
  - Up to 16 IP addresses can be assigned to a single NAT gateway.
  - A default TCP idle timeout of 4 minutes. If idle timeout is altered to a higher value, flows will be held for longer, which will increase the pressure on the SNAT port inventory.
- NAT gateway can't provide zonal isolation out-of-the-box. To get zonal redundancy, a subnet containing zonal resources must be aligned with corresponding zonal NAT gateways.

Design recommendations

- Minimize the number of outgoing Internet connections as this will impact NAT performance.
  - If large numbers of internet-bound connections are required, consider using Azure NAT Gateway to abstract outbound traffic flows.
- Use Azure Firewall where requirements to control and inspect outbound internet traffic exist.
  - Ensure Azure Firewall isn't used to inspect traffic between Azure services.

> **Note**
>
> When deploying within an Azure landing zone, consider using the foundational platform Azure Firewall resource (or equivalent NVA). If a dependency is taken on a central platform resource for internet egress, then the reliability level of that

resource and associated network path should be closely aligned with application requirements. Operational data from the resource should also be made available to the application in order to inform potential operational action in failure scenarios.

If there are high-scale requirements associated with outbound traffic, consideration should be given to a dedicated Azure Firewall resource for a mission-critical application, to mitigate risks associated with using a centrally shared resource, such as noisy neighbor scenarios.

- When deployed within a Virtual WAN environment, consideration should be given to Firewall Manager to provide centralized management of dedicated application Azure Firewall instances to ensure organizational security postures are observed through global firewall policies.
- Ensure incremental firewall policies are delegated to application security teams via role-based access control to allow for application policy autonomy.

Inter-zone and inter-region Connectivity

While the application design strongly advocates independent regional deployment stamps, many application scenarios may still require network integration between application components deployed within different zones or Azure regions, even if only under degraded service circumstances. The method by which inter-zone and inter-region communication is achieved has a significant bearing on overall performance and reliability, which will be explored through the considerations and recommendations within this section.

Design Considerations

- The application design approach for a mission-critical application endorses the use of independent regional deployments with zonal redundancy applied at all component levels within a single region.

- An Availability Zone (AZ) is a physically separate data center location within an Azure region, providing physical and logical fault isolation up to the level of a single data center.

  A round-trip latency of less than 2 ms is guaranteed for inter-zone communication. Zones will have a small latency variance given varied distances and fiber paths between zones.

- Availability Zone connectivity depends on regional characteristics, and therefore traffic entering a region via an edge location may need to be routed between zones to reach its destination. This will add a ~1ms-2ms latency given inter-zone routing and 'speed of light' constraints, but this should only have a bearing for hyper sensitive workloads.

- Availability Zones are treated as logical entities within the context of a single subscription, so different subscriptions might have a different zonal mapping for the same region. For example, zone 1 in Subscription A could correspond to the same physical data center as zone 2 in subscription B.

- Communication between zones within a region incurs a data transfer charge per GB of bandwidth.

- With application scenarios that are extremely chatty between application components, spreading application tiers across zones can introduce significant latency and increased costs. It's possible to mitigate this within the design by constraining a deployment stamp to a single zone and deploying multiple stamps across the different zones.

- Communication between different Azure regions incurs a larger data transfer charge per GB of bandwidth.
  - The applicable data transfer rate depends on the continent of the considered Azure regions.
  - Data traversing continents are charged at a considerably higher rate.

- Express Route and VPN connectivity methods can also be used to directly connect different Azure regions together for certain scenarios, or even different cloud platforms.

- For services to service communication Private Link can be used for direct communication using private endpoints.

- Traffic can be hair-pinned through Express Route circuits used for on-premise connectivity in order to facilitate routing between virtual networks within an Azure region and across different Azure regions within the same geography.
  - Hair-pinning traffic through Express Route will bypass data transfer costs associated with virtual network peering, so can be used as a way to optimize costs.
  - This approach necessitates additional network hops for application integration within Azure, which introduces latency and reliability risks. Expands the role of Express Route and associated gateway components from Azure/on-premises to also encompass Azure/Azure connectivity.

- When submillisecond latency are required between services, Proximity Placement Groups can be used when supported by the services used.

Design recommendations

- Use virtual network peering to connect networks within a region and across different regions. It's strongly recommended to avoid hair-pinning within Express Route.

- Use Private Link to establish communication directly between services in the same region or across regions (service in Region A communicating with service in Region B.

- For application workloads that are extremely chatty between components, consider constraining a deployment stamp to a single zone and deploying multiple stamps across the different zones. This ensures zonal redundancy is maintained at the level of an encapsulated deployment stamp rather than a single application component.

- Where possible, treat each deployment stamp as independent and disconnected from other stamps.
    - Use data platform technologies to synchronize state across regions rather than achieving consistency at an application level with direct network paths.
    - Avoid 'dog legging' traffic between different regions unless necessary, even in a failure scenario. Use global routing services and end-to-end health probes to take an entire stamp out of circulation in the event that a single critical component tier fails, rather than routing traffic at that faulty component level to another region.

- For hyper latency sensitive application scenarios, prioritize the use of zones with regional network gateways to optimize network latency for ingress paths.

Micro-segmentation and Kubernetes network policies

Micro-segmentation is a network security design pattern used to isolate and secure individual application workloads, with policies applied to limit network traffic between workloads based on a Zero Trust model. It's typically applied to reduce network attack surface, improve breach containment, and strengthen security through policy-driven application-level network controls.

A mission-critical application can enforce application-level network security using Network Security Groups (NSG) at either a subnet or network interface level, service Access Control Lists (ACL), and network policies when using Azure Kubernetes Service (AKS).

This section explores the optimal use of these capabilities, providing key considerations and recommendations to achieve application-level micro-segmentation.

Design Considerations

- AKS can be deployed in two different networking models:
  - **Kubenet networking:** AKS nodes are integrated within an existing virtual network, but pods exist within a virtual overlay network on each node. Traffic between pods on different nodes is routed through kube-proxy.
  - **Azure Container Networking Interface (CNI) networking:** The AKS cluster is integrated within an existing virtual network and its nodes, pods and services received IP addresses from the same virtual network the cluster nodes are attached to. This is relevant for various networking scenarios requiring direct connectivity from and to pods. Different node pools can be deployed into different subnets.

  > ⓘ **Note**
  >
  > Azure CNI requires more IP address space compared to Kubenet. Proper upfront planning and sizing of the network is required. For more information, refer to the **Azure CNI documentation**.

- By default, pods are non-isolated and accept traffic from any source and can send traffic to any destination; a pod can communicate with every other pod in a given Kubernetes cluster; Kubernetes doesn't ensure any network level isolation, and doesn't isolate namespaces at the cluster level.

- Communication between Pods and Namespaces can be isolated using Network Policies. Network Policy is a Kubernetes specification that defines access policies for communication between Pods. Using Network Policies, an ordered set of rules can be defined to control how traffic is sent/received, and applied to a collection of pods that match one or more label selectors.
  - AKS supports two plugins that implement Network Policy, *Azure* and *Calico*. Both plugins use Linux IPTables to enforce the specified policies. See Differences between Azure and Calico policies and their capabilities for more details.
  - Network policies don't conflict since they're additive.

- For a network flow between two pods to be allowed, both the egress policy on the source pod and the ingress policy on the destination pod need to allow the traffic.
    - The network policy feature can only be enabled at cluster instantiation time. It is not possible to enable network policy on an existing AKS cluster.
    - The delivery of network policies is consistent regardless of whether Azure or Calico is used.
    - Calico provides a richer feature set, including support for windows-nodes and supports Azure CNI as well as Kubenet.

- AKS supports the creation of different node pools to separate different workloads using nodes with different hardware and software characteristics, such as nodes with and without GPU capabilities.
    - Using node pools doesn't provide any network-level isolation.
    - Node pools can use different subnets within the same virtual network. NSGs can be applied at the subnet-level to implement micro-segmentation between node pools.

Design recommendations

- Configure an NSG on all considered subnets to provide an IP ACL to secure ingress paths and isolate application components based on a Zero Trust model.

    - Use Front Door Service Tags within NSGs on all subnets containing application backends defined within Azure Front Door, since this will validate traffic originates from a legitimate Azure Front Door backend IP address space. This will ensure traffic flows through Azure Front Door at a service level, but header based filtering will still be required to ensure the use of a particular Front Door instance and to also mitigate 'IP spoofing' security risks.

    - Public internet traffic should be disabled on RDP and SSH ports across all applicable NSGs.

    - Prioritize the use of the Azure CNI network plugin and consider Kubenet for scenarios with a limited range of available IP addresses to fit the application within a constrained address space.
        - AKS supports the use of both Azure CNI and Kubenet. It is selected at deployment time.
        - The Azure CNI network plugin is a more robust and scalable network plugin, and is recommended for most scenarios.
        - Kubenet is a more lightweight network plugin, and is recommended for scenarios with a limited range of available IP addresses.

- See Azure CNI for more details.

- The Network Policy feature in Kubernetes should be used to define rules for ingress and egress traffic between pods in a cluster. Define granular Network Policies to restrict and limit cross-pod communication.
  - Enable Network Policy for Azure Kubernetes Service at deployment time.
  - Prioritize the use of *Calico* because it provides a richer feature set with broader community adoption and support.

Next step

Review the considerations for quantifying and observing application health.

Health modeling and observability

Health modeling and observability of mission-critical workloads on Azure

Article • 02/01/2023

Health modeling and observability are essential concepts to maximize reliability, which focuses on robust and contextualized instrumentation and monitoring. These concepts provide critical insight into application health, promoting the swift identification and resolution of issues.

Most mission-critical applications are significant in terms of both scale and complexity and therefore generate high volumes of operational data, which makes it challenging to evaluate and determine optimal operational action. Health modeling ultimately strives to maximize observability by augmenting raw monitoring logs and metrics with key business requirements to quantify application health, driving automated evaluation of health states to achieve consistent and expedited operations.

This design area focuses on the process to define a robust health model, mapping quantified application health states through observability and operational constructs to achieve operational maturity.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**

There are three main levels of operational maturity when striving to maximize reliability.

1. *Detect* and respond to issues as they happen.
2. *Diagnose* issues that are occurring or have already occurred.
3. *Predict* and prevent issues before they take place.

Video: Define a health model for your mission-critical workload

https://learn-video.azurefd.net/vod/player?id=fd8c4e50-9d7f-4df0-97cb-d0474b581398&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-health-modeling&locale=en-us

Layered application health

To build a health model, first define application health in the context of key business requirements by quantifying ‘healthy’ and ‘unhealthy’ states in a layered and measurable format. Then, for each application component, refine the definition in the context of a steady running state and aggregated according to the application user flows. Superimpose with key non-functional business requirements for performance and availability. Finally, aggregate the health states for each individual user flow to form an acceptable representation of the overall application health. Once established, these layered health definitions should be used to inform critical monitoring metrics across all system components and validate operational subsystem composition.

> **Important**
>
> When defining what 'unhealthy' states, represent for all levels of the application. It's important to distinguish between transient and non-transient failure states to qualify service degradation relative to unavailability.

Design considerations

- The process of modeling health is a top-down design activity that starts with an architectural exercise to define all user flows and map dependencies between functional and logical components, thereby implicitly mapping dependencies between Azure resources.

- A health model is entirely dependent on the context of the solution it represents, and therefore can't be solved 'out-of-the-box' because 'one size doesn't fit all'.
  - Applications will differ in composition and dependencies
  - Metrics and metric thresholds for resources must also be finely tuned in terms of what values represent healthy and unhealthy states, which are heavily influenced by encompassed application functionality and target non-functional requirements.

- A layered health model enables application health to be traced back to lower level dependencies, which helps to quickly root cause service degradation.

- To capture health states for an individual component, that component's distinct operational characteristics must be understood under a steady state that is reflective of production load. Performance testing is therefore a key capability to define and continually evaluate application health.

- Failures within a cloud solution may not happen in isolation. An outage in a single component may lead to several capabilities or additional components becoming unavailable.
  - Such errors may not be immediately observable.

Design recommendations

- Define a measurable health model as a priority to ensure a clear operational understanding of the entire application.
  - The health model should be layered and reflective of the application structure.
  - The foundational layer should consider individual application components, such as Azure resources.
  - Foundational components should be aggregated alongside key non-functional requirements to build a business-contextualized lens into the health of system flows.
  - System flows should be aggregated with appropriate weights based on business criticality to build a meaningful definition of overall application health. Financially significant or customer-facing user flows should be prioritized.
  - Each layer of the health model should capture what 'healthy' and 'unhealthy' states represent.
  - Ensure the heath model can distinguish between transient and non-transient unhealthy states to isolate service degradation from unavailability.

- Represent health states using a granular health score for every distinct application component and every user flow by aggregating health scores for mapped dependent components, considering key non-functional requirements as coefficients.
  - The health score for a user flow should be represented by the lowest score across all mapped components, factoring in relative attainment against non-functional requirements for the user flow.
  - The model used to calculate health scores must consistently reflect operating health, and if not, the model should be adjusted and redeployed to reflect new learnings.
  - Define health score thresholds to reflect health status.

- The health score must be calculated automatically based on underlying metrics, which can be visualized through observability patterns and acted on through automated operational procedures.
  - The health score should become core to the monitoring solution, so that operating teams no longer have to interpret and map operational data to application health.

- Use the health model to calculate availability Service Level Objective (SLO) attainment instead of raw availability, ensuring the demarcation between service degradation and unavailability is reflected as separate SLOs.

- Use the health model within CI/CD pipelines and test cycles to validate application health is maintained after code and configuration updates.
  - The health model should be used to observe and validate health during load testing and chaos testing as part of CI/CD processes.

- Building and maintaining a health model is an iterative process and engineering investment should be aligned to drive continuous improvements.
  - Define a process to continually evaluate and fine-tune the accuracy of the model, and consider investing in machine learning models to further train the model.

Example - Layered health model

This is a simplified representation of a layered application health model for illustrative purposes. A comprehensive and contextualized health model is provided in the Mission-Critical reference implementations:

- Mission-Critical Online
- Mission-Critical Connected

When implementing a health model it's important to define the health of individual components through the aggregation and interpretation of key resource-level metrics. An example of how resource metrics can be used is the image below:

	EventProcessor	Rest API	K8s Cluster	Storage
All Good	Queue Depth < 10 Processing Time < 100ms Time in Queue < 200ms	Response Time < 150ms Failed Request Count < 10	Avg_CPU% < 75 Autoscaled% < 60 Pod_Restart_Count< 1 Pod_Avg_CPU_Util% < 80	Response Time < 100ms Request_Failure_Count < 2
Attention	Queue Depth < 50 Processing Time < 200ms Time in Queue < 1000ms	Response Time < 300ms Failed Request Count < 50	Avg CPU% < 90 Autoscaled% < 90 Pod_Restart_Count < 5 Pod_Avg_CPU_Util% < 90	Response Time < 200ms Request_Failure_Count < 5
User impact	Queue Depth > 50 Processing Time > 200ms Time in Queue > 1000ms	Response Time > 300ms Failed Request Count > 50	Avg CPU% > 90 Autoscaled% >90 Pod_Restart_Count > 5 Pod_Avg_CPU_Util% > 90	Response Time > 200ms Request_Failure_Count > 5

This definition of health can subsequently be represented by a KQL query, as demonstrated by the example AKS query below that aggregates InsightsMetrics (AKS

Container insights) and AzureMetrics (Azure diagnostics) and compares (inner join) against modeled health thresholds.

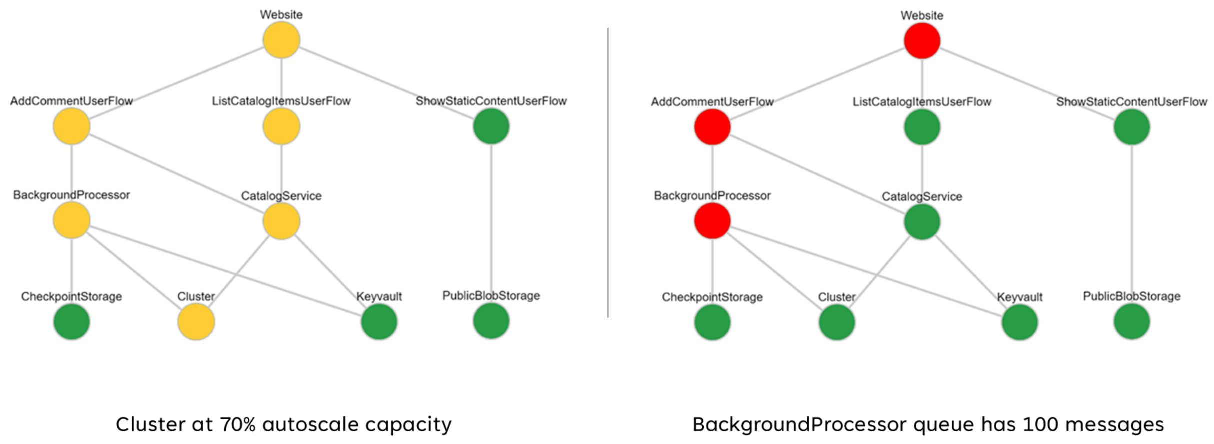
```kql
// ClusterHealthStatus
let Thresholds=datatable(MetricName: string, YellowThreshold: double,
RedThreshold: double) [
    // Disk Usage:
    "used_percent", 50, 80,
    // Average node cpu usage %:
    "node_cpu_usage_percentage", 60, 90,
    // Average node disk usage %:
    "node_disk_usage_percentage", 60, 80,
    // Average node memory usage %:
    "node_memory_rss_percentage", 60, 80
    ];
InsightsMetrics
| summarize arg_max(TimeGenerated, *) by Computer, Name
| project TimeGenerated,Computer, Namespace, MetricName = Name, Value=Val
| extend NodeName = extract("([a-z0-9-]*)(-)([a-z0-9]*)$", 3, Computer)
| union (
    AzureMetrics
    | extend ResourceType = extract("(PROVIDERS/MICROSOFT.)([A-Z]*/[A-Z]*)",
2, ResourceId)
    | where ResourceType == "CONTAINERSERVICE/MANAGEDCLUSTERS"
    | summarize arg_max(TimeGenerated, *) by MetricName
    | project TimeGenerated, MetricName, Namespace = "AzureMetrics",
Value=Average
    )
| lookup kind=inner Thresholds on MetricName
| extend IsYellow = iff(Value > YellowThreshold and Value < RedThreshold, 1,
0)
| extend IsRed = iff(Value > RedThreshold, 1, 0)
| project NodeName, MetricName, Value, YellowThreshold, IsYellow,
RedThreshold, IsRed
```

The resulting table output can subsequently be transformed into a health score for easier aggregation at higher levels of the health model.

```kql
// ClusterHealthScore
ClusterHealthStatus
| summarize YellowScore = max(IsYellow), RedScore = max(IsRed)
| extend HealthScore = 1-(YellowScore*0.25)-(RedScore*0.5)
```

These aggregated scores can subsequently be represented as a dependency chart using visualization tools such as Grafana to illustrate the health model.

This image shows an example layered health model from the Azure Mission-Critical online reference implementation, and demonstrates how a change in health state for a foundational component can have a cascading impact to user flows and overall application health (the example values correspond to the table in the previous image).



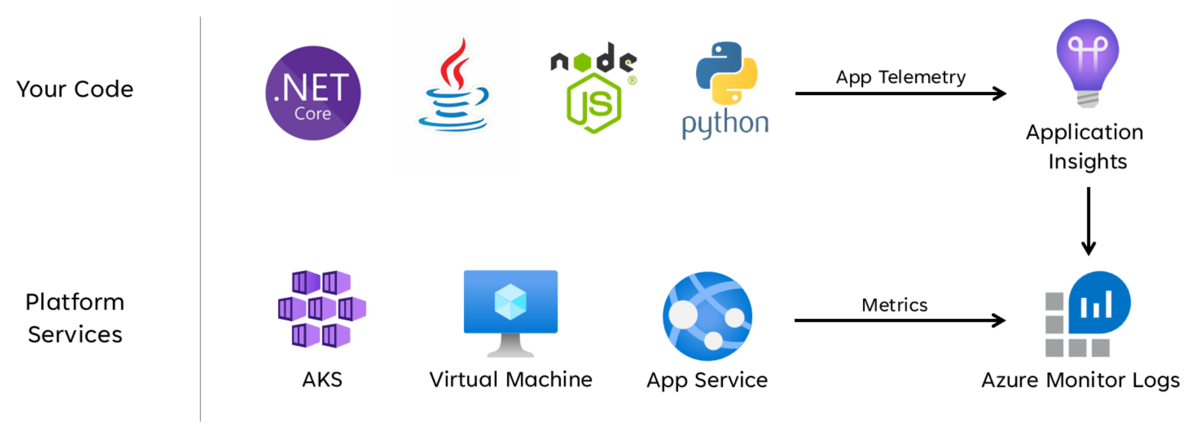
Demo video: Monitoring and health modeling demo

https://www.microsoft.com/en-us/videoplayer/embed/RE55Nd9?postJslIMsg=true

Unified data sink for correlated analysis

Many operational datasets must be gathered from all system components to accurately represent a defined heath model, considering logs and metrics from both application components and underlying Azure resources. This vast amount of data ultimately needs to be stored in a format that allows for near-real time interpretation to facilitate swift operational action. Moreover, correlation across all encompassed data sets is required to ensure effective analysis is unbounded, allowing for the layered representation of health.

A unified data sink is required to ensure all operational data is swiftly stored and made available for correlated analysis to build a 'single pane' representation of application health. Azure provides several different operational technologies under the umbrella of Azure Monitor, and Azure Monitor Log Analytics serves as the core Azure-native data sink to store and analyze operational data.



Design considerations

Azure Monitor

- Azure Monitor is enabled by default for all Azure subscriptions, but Azure Monitor for Logs (Log Analytics) and Azure Application Insights resources must be deployed and configured to incorporate data collection and querying capabilities.
- Azure Monitor supports three types of observability data: logs, metrics, and distributed traces.
  - Logs are stored in Azure Monitor Logs workspaces based on Azure Data Explorer. Log queries are stored in query packs that can be shared across subscriptions, and are used to drive observability components such as dashboards, workbooks, or other reporting and visualization tools.
  - Metrics are stored in an internal time-series diagnostic service database. For most Azure resources, the retention period is retained for 93 days. Metric collection is configured through resource Diagnostic settings.
- All Azure resources expose logs and metrics, but resources must be appropriately configured to route diagnostic data to your desired data sink.

> **Tip**
>
> Azure provides various **Built-In Policies** that can be applied to ensure deployed resources are configured to send logs and metrics to an Azure Monitor instance.

- It's not uncommon for regulatory controls to require operational data remains within originating geographies or countries. Regulatory requirements may stipulate the retention of critical data types for an extended period of time. For example, in regulated banking, audit data must be retained for at least seven years.

- Different operational data types may require different retention periods. For example, security logs may need to be retained for a long period, while performance data is unlikely to require long-term retention outside the context of AIOps.
- Data can be exported from Log Analytics Workspaces for long term retention and/or auditing purposes.
- Azure Monitor Logs Dedicated Clusters provides a deployment option that enables Availability Zones for protection from zonal failures in supported Azure regions. Dedicated Clusters require a minimum daily data ingest commitment.
- Azure Monitor for Logs resources, including underlying log and metrics storage, are deployed into a specified Azure region.
- To protect against loss of data from unavailability of an Azure Monitor for Logs workspace, resources can be configured with multiple Diagnostics configurations. Each Diagnostic configuration can target metrics and logs at a separate Azure Monitor for Log workspace.
  - Each additional Azure Monitor for Logs workspace will incur extra costs.
  - The redundant Azure Monitor for Logs workspaces can be deployed into the same Azure region, or into separate Azure regions for additional regional redundancy.
  - Sending logs and metrics from an Azure resource to an Azure Monitor for Logs workspace in a different region will incur inter-region data egress costs.
  - Some Azure resources require an Azure Monitor for Logs workspace within the same region as the resource itself.
- Azure Monitor Logs workspace data can be exported to Azure Storage or Azure Event Hubs on a continuous, scheduled, or one-time basis.
  - Data export allows for long-term data archiving and protects against possible operational data loss due to unavailability.
  - Available export destinations are Azure Storage or Azure Event Hubs. Azure Storage can be configured for different redundancy levels including zonal or regional. Data export to Azure Storage stores the data within .json files.
  - Data export destinations must be within the same Azure region as the Azure Monitor Logs workspace. An event hub data export destination to be within the same region as the Azure Monitor Logs workspace. Azure Event Hubs geo-disaster recovery isn't applicable for this scenario.
  - There are several data export limitations. Only specific Azure Monitor Logs tables are supported for data export.

- Azure Monitor Logs has user query throttling limits, which may appear as reduced availability to clients, such as observability dashboards.
  - Five concurrent queries per user: if five queries are already running, additional queries are placed in a per-user concurrency queue until a running query ends.
  - Time in concurrency queue: if a query sits in the concurrency queue for over three minutes, it will be terminated and a 429 error code returned.
  - Concurrency queue depth limit: the concurrency queue is limited to 200 queries, and additional queries will be rejected with a 429 error code.
  - Query rate limit: there's a per-user limit of 200 queries per 30 seconds across all workspaces.

- Query Packs are Azure Resource Manager resources, which can be used to protect and recover Azure Monitor Logs queries if Azure Monitor Logs workspace is unavailable.
  - Query Packs contain queries as JSON and can be stored external to Azure similar to other infrastructure-as-code assets.
    - Deployable through the Microsoft.Insights REST API.
    - If an Azure Monitor for Logs workspace must be re-created the Query Pack can be redeployed from an externally stored definition.

- Application Insights can be deployed in a workspace-based deployment model, underpinned by a Log Analytics Workspace where all the data is stored.

- Sampling can be enabled within Application Insights to reduce the amount of telemetry sent and optimize data ingest costs.

- Log Analytics and Application Insights charge based on the volume of data ingested and the duration that data is retained for.
  - Data ingested into a Log Analytics Workspace can be retained at no additional charge up to first 31 days (90 days if Sentinel is enabled)
  - Data ingested into a Workspace-based Application Insights is retained for the first 90 days at no extra charge.

- The Log Analytics Commitment Tier pricing model provides a predictable approach to data ingest charges.
  - Any usage above the reservation level is billed at the same price as the current tier.

- Azure Monitor Log Analytics, Application Insights, and Azure Data Explorer use the Kusto Query Language (KQL).

- Log Analytics queries are saved as *functions* within Log Analytics (`savedSearches`).

Design recommendations

- Use Azure Monitor for Logs (Log Analytics) as a unified data sink to provide a 'single pane' across all operational data sets.
    - Decentralize Log Analytics Workspaces across all used deployment regions. Each Azure region with an application deployment should consider a Log Analytics Workspace to gather all operational data originating from that region. All global resources should use a separate dedicated Log Analytics Workspace, which should be deployed within a primary deployment region.
        - Sending all operational data to a single Log Analytics Workspace would create a single point of failure.
        - Requirements for data residency might prohibit data leaving the originating region, and federated workspaces solves for this requirement by default.
        - There's a substantial egress cost associated with transferring logs and metrics across regions.
    - All deployment stamps within the same region can use the same regional Log Analytics Workspace.

- Consider configuring resources with multiple diagnostic configurations pointing to different Azure Monitor for Logs workspaces to protect against Azure Monitor unavailability for applications with fewer regional deployment stamps.

- Use Application Insights as a consistent Application Performance Monitoring (APM) tool across all application components to collect application logs, metrics, and traces.
    - Deploy Application Insights in a workspace-based configuration to ensure each regional Log Analytics Workspaces contains logs and metrics from both application components and underlying Azure resources.

- Use Cross-Workspace queries to maintain a unified 'single pane' across the different workspaces.

- Use Query Packs to protect Azure Monitor Logs queries in the event of workspace unavailability.
    - Store query packs within the application git repository as infrastructure-as-code assets.

- All Log Analytics Workspaces should be treated as long-running resources with a different life-cycle to application resources within a regional deployment stamp.

- Export critical operational data from Log Analytics for long-term retention and analytics to facilitate AIOps and advanced analytics to refine the underlying health model and inform predictive action.

- Carefully evaluate which data store should be used for long-term retention; not all data has to be stored in a hot and queryable data store.
  - It's strongly recommended to use Azure Storage in a GRS configuration for long-term operational data storage.
    - Use the Log Analytics Export capability to export all available data sources to Azure Storage.

- Select appropriate retention periods for operational data types within log analytics, configuring longer retention periods within the workspace where 'hot' observability requirements exist.

- Use Azure Policy to ensure all regional resources route operational data to the correct Log Analytics Workspace.

> **Note**
>
> When deploying into an Azure landing zone, if there's a requirement for centralized storage of operational data, you can **fork** data at instantiation so it's ingested into both centralized tooling and Log Analytics Workspaces dedicated to the application. Alternatively, expose access to application Log Analytics workspaces so that central teams can query application data. It's ultimately critical that operational data originating from the solution is available within Log Analytics Workspaces dedicated to the application.
>
> If SIEM integration is required, do not send raw log entries, but instead send critical alerts.

- Only configure sampling within Application Insights if it's required to optimize performance, or if not sampling becomes cost prohibitive.
  - Excessive sampling can lead to missed or inaccurate operational signals.

- Use correlation IDs for all trace events and log messages to tie them to a given request.
  - Return correlation IDs to the caller for all calls not just failed requests.

- Ensure application code incorporates proper instrumentation and logging to inform the health model and facilitate subsequent troubleshooting or root cause analysis when required.
  - Application code should use Application Insights to facilitate Distributed Tracing, by providing the caller with a comprehensive error message that includes a correlation ID when a failure occurs.

- Use structured logging for all log messages.

- Add meaningful health probes to all application components.
  - When using AKS, configure the health endpoints for each deployment (pod) so that Kubernetes can correctly determine when a pod is healthy or unhealthy.
  - When using Azure App Service, configure the Health Checks so that scale out operations will not cause errors by sending traffic to instances that are not-yet ready, and making sure unhealthy instances are recycled quickly.

> If the application is subscribed to Microsoft Mission-Critical Support, consider exposing key health probes to Microsoft Support, so application health can be modelled more accurately by Microsoft Support.

- Log successful health check requests, unless increased data volumes can't be tolerated in the context of application performance, since they provide additional insights for analytical modeling.

- Do not configure production Log Analytics Workspaces to apply a daily cap, which limits the daily ingestion of operational data, since this can lead to the loss of critical operational data.
  - In lower environments, such as Development and Test, it can be considered as an optional cost saving mechanism.

- Provided operational data ingest volumes meet the minimum tier threshold, configure Log Analytics Workspaces to use Commitment Tier based pricing to drive cost efficiencies relative to the 'pay-as-you-go' pricing model.

- It's strongly recommended to store Log Analytics queries using source control and use CI/CD automation to deploy them to relevant Log Analytics instances.

Visualization

Visually representing the health model with critical operational data is essential to achieve effective operations and maximize reliability. Dashboards should ultimately be utilized to provide near-real time insights into application health for DevOps teams, facilitating the swift diagnosis of deviations from steady state.

Microsoft provides several data visualization technologies, including Azure Dashboards, Power BI, and Azure Managed Grafana (currently in-preview). Azure Dashboards is positioned to provide a tightly integrated out-of-the-box visualization solution for operational data within Azure Monitor. It therefore has a fundamental role to play in the visual representation of operational data and application health for a mission-critical workload. However, there are several limitations in terms of the positioning of Azure Dashboards as a holistic observability platform, and as a result consideration should be given to the supplemental use of market-leading observability solutions, such as Grafana, which is also provided as a managed solution within Azure.

This section focuses on the use of Azure Dashboards and Grafana to build a robust dashboarding experience capable of providing technical and business lenses into application health, enabling DevOps teams and effective operation. Robust dashboarding is essential to diagnose issues that have already occurred, and support operational teams in detecting and responding to issues as they happen.

Design considerations

- When visualizing the health model using Log Analytics queries, note that there are Log Analytics limits on concurrent and queued queries, as well as the overall query rate, with subsequent queries queued and throttled.

- Queries to retrieve operational data used to calculate and represent health scores can be written and executed in either Azure Monitor Log Analytics or Azure Data Explorer.
  - Sample queries are available here.

- Log Analytics imposes several query limits, which must be designed for when designing operational dashboards.

- The visualization of raw resource metrics, such as CPU utilization or network throughput, requires manual evaluation by operations teams to determine health status impacts, and this can be challenging during an active incident.

- If multiple users use dashboards within a tool like Grafana, the number of queries sent to Log Analytics multiplies quickly.
  - Reaching the concurrent query limit on Log Analytics will queue subsequent queries, making the dashboard experience feel 'slow'.

Design Recommendations

- Collect and present queried outputs from all regional Log Analytics Workspaces and the global Log Analytics Workspace to build a unified view of application health.

> ⓘ **Note**
>
> If deploying into an Azure landing zone, consider querying the **central platform Log Analytics Workspace** if key dependencies on platform resources exist, such as

ExpressRoute for on-premises communication.

- A ‘traffic light’ model should be used to visually represent 'healthy' and 'unhealthy' states, with green used to illustrate when key non-functional requirements are fully satisfied and resources are optimally utilized. Use "Green", "Amber, and "Red" to represent "Healthy", "Degraded", and "Unavailable" states.

- Use Azure Dashboards to create operational lenses for global resources and regional deployment stamps, representing key metrics such as request count for Azure Front Door, server side latency for Azure Cosmos DB, incoming/outgoing messages for Event Hubs, and CPU utilization or deployment statuses for AKS. Dashboards should be tailored to drive operational effectiveness, infusing learnings from failure scenarios to ensure DevOps teams have direct visibility into key metrics.

- If Azure Dashboards can't be used to accurately represent the health model and requisite business requirements, then it's strongly recommended to consider Grafana as an alternative visualization solution, providing market-leading capabilities and an extensive open-source plugin ecosystem. Evaluate the managed Grafana preview offering to avoid the operational complexities of managing Grafana infrastructure.

- When deploying self-hosted Grafana, employ a highly available and geo-distributed design to ensure critical operational dashboards can be resilient to regional platform failures and cascading error scenarios.

  - Separate configuration state into an external datastore, such as Azure Database for Postgres or MySQL, to ensure Grafana application nodes remain stateless.
    - Configure database replication across deployment regions.

  - Deploy Grafana nodes to App Services in a highly available configuration across ones within a region, using container based deployments.
    - Deploy App Service instances across considered deployment regions.

  > App Services provides a low-friction container platform, which is ideal for low-scale scenarios such as operational dashboards, and isolating Grafana from AKS provides a clear separation of concern between the primary application platform and operational representations for that platform. Please refer to the Application Platform deign area for further configuration recommendations.

  - Use Azure Storage in a GRS configuration to host and manage custom visuals and plugins.
  - Deploy app service and database read-replica Grafana components to a minimum of two deployment regions, and consider employing a model where Grafana is deployed to all considered deployment regions.

> For scenarios targeting a >= 99.99% SLO, Grafana should be deployed within a minimum of 3 deployment regions to maximize overall reliability for key operational dashboards.

- Mitigate Log Analytics query limits by aggregating queries into a single or small number of queries, such as by using the KQL 'union' operator, and set an appropriate refresh rate on the dashboard.
  - An appropriate maximum refresh rate will depend on the number and complexity of dashboard queries; analysis of implemented queries is required.
- If the concurrent query limit of log analytics is being reached, consider optimizing the retrieval pattern by (temporarily) storing the data required for the dashboard in a high performance datastore such as Azure SQL.

Automated incident response

While the visual representations of application health provide invaluable operational and business insights to support issue detection and diagnosis, it relies on the readiness and interpretations of operational teams, as well as the effectiveness of subsequent human-triggered responses. Therefore, to maximize reliability it's necessary to implement extensive alerting to detect proactively and respond to issues in near real-time.

Azure Monitor provides an extensive alerting framework to detect, categorize, and respond to operational signals through Action Groups. This section will therefore focus on the use of Azure Monitor alerts to drive automated actions in response to current or potential deviations from a healthy application state.

> ⓘ **Important**
>
> Alerting and automated action is critical to effectively detect and swiftly respond to issues as they happen, before greater negative impact can occur. Alerting also provides a mechanism to interpret incoming signals and respond to prevent issues before they occur.

Design considerations

- Alert rules are defined to fire when a conditional criteria is satisfied for incoming signals, which can include various data sources, such as metrics, log search queries, or availability tests.
- Alerts can be defined within Log Analytics or Azure Monitor on the specific resource.
- Some metrics are only interrogatable within Azure Monitor, since not all diagnostic data points are made available within Log Analytics.
- The Azure Monitor Alerts API can be used to retrieve active and historic alerts.
- There are subscription limits related to alerting and action groups, which must be designed for:
  - Limits exist for the number of configurable alert rules.
  - The Alerts API has throttling limits, which should be considered for extreme usage scenarios.
  - Action Groups have several hard limits for the number of configurable responses, which must be designed for.
    - Each response type has a limit of 10 actions, apart from email, which has a limit of 1,000 actions.
- Alerts can be integrated within a layered health model by creating an Alert Rule for a saved log search query from the model's 'root' scoring function. For example, using 'WebsiteHealthScore' and alerting on a numeric value that represents an 'Unhealthy' state.

Design recommendations

- For resource-centric alerting, create alert rules within Azure Monitor to ensure all diagnostic data is available for the alert rule criteria.
- Consolidate automated actions within a minimal number of Action Groups, aligned with service teams to support a DevOps approach.
- Respond to excessive resource utilization signals through automated scale operations, using Azure-native auto-scale capabilities where possible. Where built-in auto-scale functionality isn't applicable, use the component health score to model signals and determine when to respond with automated scale operations. Ensure automated scale operations are defined according to a capacity model,

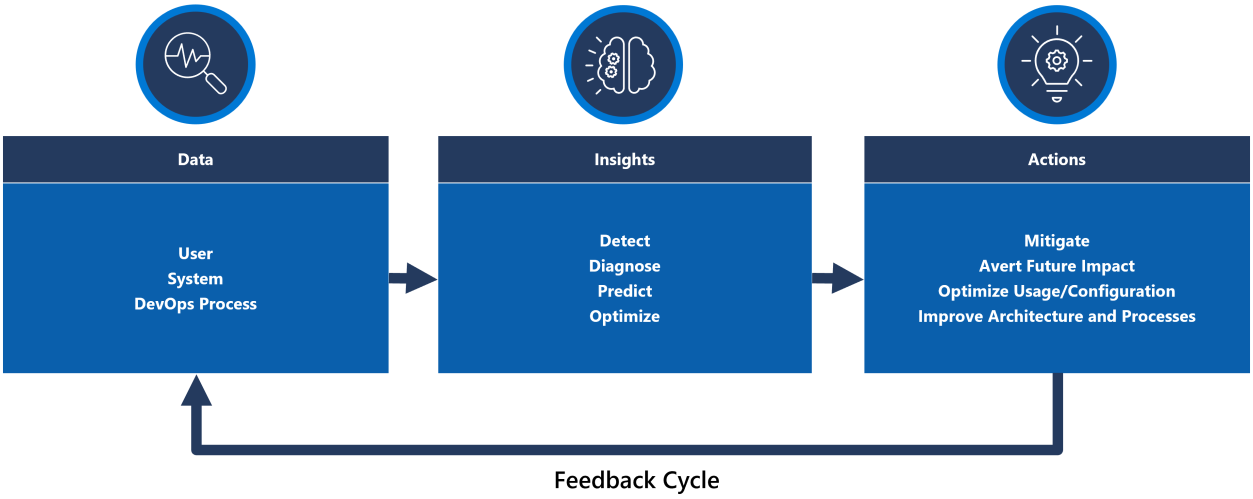
which quantifies scale relationships between components, so that scale responses encompass components that need to be scaled in relation to other components.

- Model actions to accommodate a prioritized ordering, which should be determined by business impact.
- Use the Azure Monitor Alerts API to gather historic alerts to incorporate within 'cold' operational storage for advanced analytics.
- For critical failure scenarios, which can't be met with an automated response, ensure operational 'runbook automation' is in-place to drive swift and consistent action once manual interpretation and sign out is provided. Use alert notifications to drive swift identification of issues requiring manual interpretation
- Create allowances within engineering sprints to drive incremental improvements in alerting to ensure new failure scenarios that haven't previously been considered can be fully accommodated within new automated actions.
- Conduct operational readiness tests as part of CI/CD processes to validate key alert rules for deployment updates.

Predictive action and AI operations (AIOps)

Machine learning models can be applied to correlate and prioritize operational data, helping to gather critical insights related to filtering excessive alert 'noise' and predicting issues before they cause impact, as well as accelerating incident response when they do.

More specifically, an AIOps methodology can be applied to critical insights about the behavior of the system, users, and DevOps processes. These insights can include identifying a problem happening now (*detect*), quantifying why the problem is happening (*diagnose*), or signaling what will happen in the future (*predict*). Such insights can be used to drive actions that adjust and optimize the application to mitigate active or potential issues, using key business metrics, system quality metrics, and DevOps productivity metrics, to prioritize according to business impact. Conducted actions can themselves be infused into the system through a feedback loop that further trains the underlying model to drive additional efficiencies.



There are multiple analytical technologies within Azure, such as Azure Synapse and Azure Databricks, which can be used to build and train analytical models for AIOps. This section will therefore focus on how these technologies can be positioned within an application design to accommodate AIOps and drive predictive action, focusing on Azure Synapse that reduces friction by bringing together the best of Azure's data services along with powerful new features.

AIOps is used to drive predictive action, interpreting and correlating complex operational signals observed over a sustained period in order to better respond to and prevent issues before they occur.

Design considerations

- Azure Synapse Analytics offers multiple Machine Learning (ML) capabilities.
  - ML models can be trained and run on Synapse Spark Pools with libraries including MLLib, SparkML and MMLSpark, as well as popular open-source libraries, such as Scikit Learn.
  - ML models can be trained with common data science tools like PySpark/Python, Scala, or .NET.

- Synapse Analytics is integrated with Azure ML through Azure Synapse Notebooks, which enables ML models to be trained in an Azure ML Workspace using Automated ML.

- Synapse Analytics also enables ML capabilities using Azure Cognitive Services to solve general problems in various domains, such as Anomaly Detection. Cognitive Services can be used in Azure Synapse, Azure Databricks, and via SDKs and REST APIs in client applications.

- Azure Synapse natively integrates with Azure Data Factory tools to extract, transform, and load (ETL) or ingest data within orchestration pipelines.

- Azure Synapse enables external dataset registration to data stored in Azure Blob storage or Azure Data Lake Storage.
  - Registered datasets can be used in Synapse Spark pool data analytics tasks.

- Azure Databricks can be integrated into Azure Synapse Analytics pipelines for additional Spark capabilities.
  - Synapse orchestrates reading data and sending it to a Databricks cluster, where it can be transformed and prepared for ML model training.

- Source data typically needs to be prepared for analytics and ML.
  - Synapse offers various tools to assist with data preparation, including Apache Spark, Synapse Notebooks, and serverless SQL pools with T-SQL and built-in visualizations.

- ML models that have been trained, operationalized, and deployed can be used for *batch* scoring in Synapse.
  - AIOps scenarios, such as running regression or degradation predictions in CI/CD pipelined, may require *real-time* scoring.

- There are subscription limits for Azure Synapse, which should be fully understood in the context of an AIOps methodology.

- To fully incorporate AIOps it's necessary to feed near real-time observability data into real-time ML inference models on an ongoing basis.
  - Capabilities such as anomaly detection should be evaluated within the observability data stream.

Design recommendations

- Ensure all Azure resources and application components are fully instrumented so that a complete operational dataset is available for AIOps model training.

- Ingest Log Analytics operational data from the global and regional Azure Storage Accounts into Azure Synapse for analysis.

- Use the Azure Monitor Alerts API to retrieve historic alerts and store it within cold storage for operational data to subsequently use within ML models. If Log Analytics data export is used, store historic alerts data in the same Azure Storage accounts as the exported Log Analytics data.

- After ingested data is prepared for ML training, write it back out to Azure Storage so that it's available for ML model training without requiring Synapse data preparation compute resources to be running.
- Ensure ML model operationalization supports both batch and real-time scoring.
- As AIOps models are created, implement MLOps and apply DevOps practices to automate the ML lifecycle for training, operationalization, scoring, and continuous improvement. Create an iterative CI/CD process for AIOps ML models.
- Evaluate Azure Cognitive Services for specific predictive scenarios due to their low administrative and integration overhead. Consider Anomaly Detection to quickly flag unexpected variances in observability data streams.

Next step

Review the deployment and testing considerations.

Deployment and testing

Deployment and testing for mission-critical workloads on Azure

Article • 02/01/2023

Failed deployments and erroneous releases are common causes for application outages. Your approach to deployment and testing plays a critical role in the overall reliability of a mission-critical application.

Deployment and testing should form the basis for all application and infrastructure operations to ensure consistent outcomes for mission-critical workloads. Be prepared to deploy weekly, daily, or more often. Design your continuous integration and continuous deployment (CI/CD) pipelines to support those goals.

The strategy should implement:

- **Rigorous pre-release testing**. Updates shouldn't introduce defects, vulnerabilities, or other factors that might jeopardize application health.

- **Transparent deployments**. It should be possible to roll out updates at any time without affecting users. Users should be able to continue their interactions with the application without interruption.

- **Highly available operations**. Deployment and testing processes and tools must be highly available to support overall application reliability.

- **Consistent deployment processes**. The same application artifacts and processes should be used to deploy the infrastructure and application code across different environments. End-to-end automation is mandatory. Manual interventions must be avoided because they can introduce reliability risks.

This design area provides recommendations on how to optimize deployment and testing processes with the goal of minimizing downtime and maintaining application health and availability.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected Framework mission-critical workload** series. If you aren't familiar with this series, we recommend that you start with **What is a mission-critical workload?**.

Zero-downtime deployment

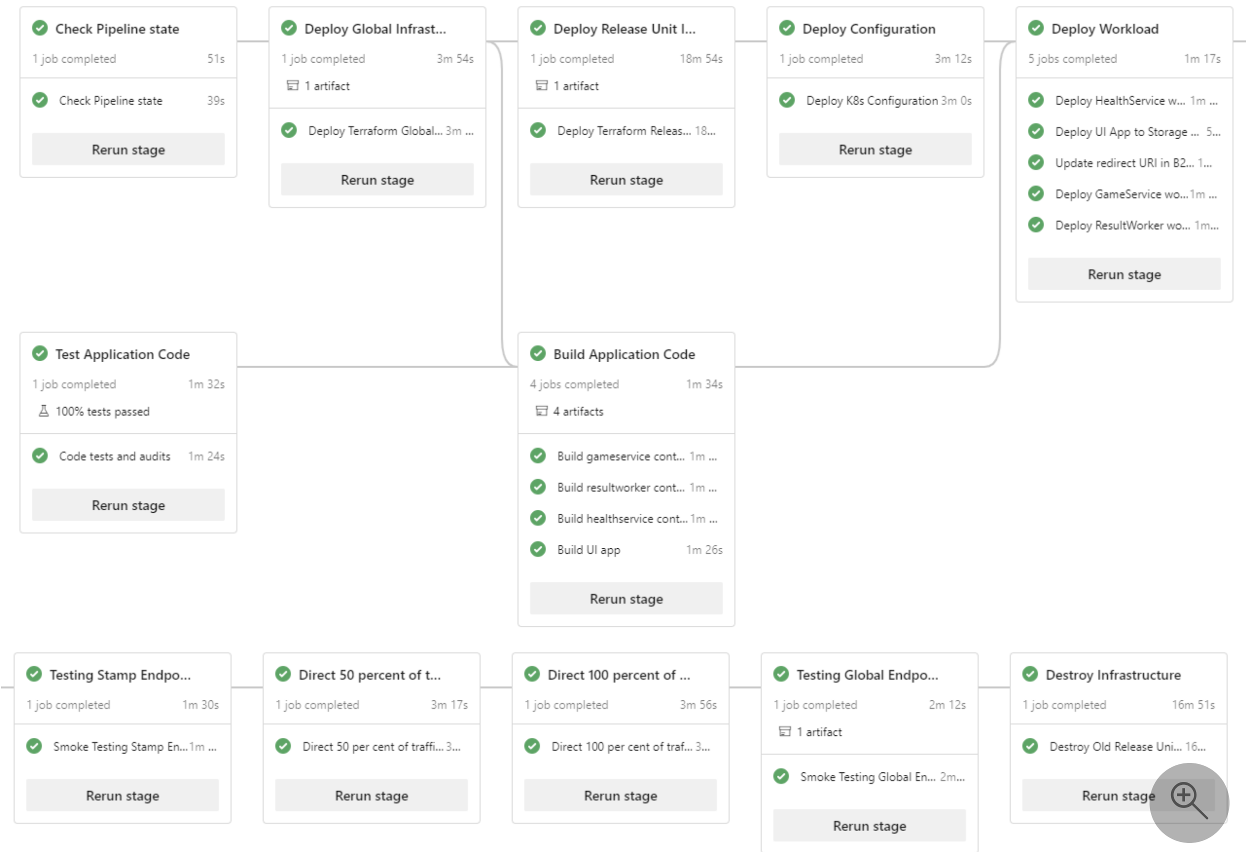
View the following video for an overview of zero-downtime deployment.

https://learn-video.azurefd.net/vod/player?id=1d3846c0-f301-4a25-a49a-94f5be3f6605&locale=en-us&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-deployment-testing

Achieving zero-downtime deployments is a fundamental goal for mission-critical applications. Your application needs to be available all day, every day, even when new releases are rolled out during business hours. Invest your efforts up front to define and plan deployment processes in order to drive key design decisions like whether to treat resources as ephemeral.

To achieve zero-downtime deployment, deploy new infrastructure next to the existing infrastructure, test it thoroughly, transition end user traffic, and only then decommission the previous infrastructure. Other practices, like the scale-unit architecture, are also key.

The Mission-Critical Online and Azure Mission-Critical Connected reference implementations illustrate this deployment approach, as shown in this diagram:

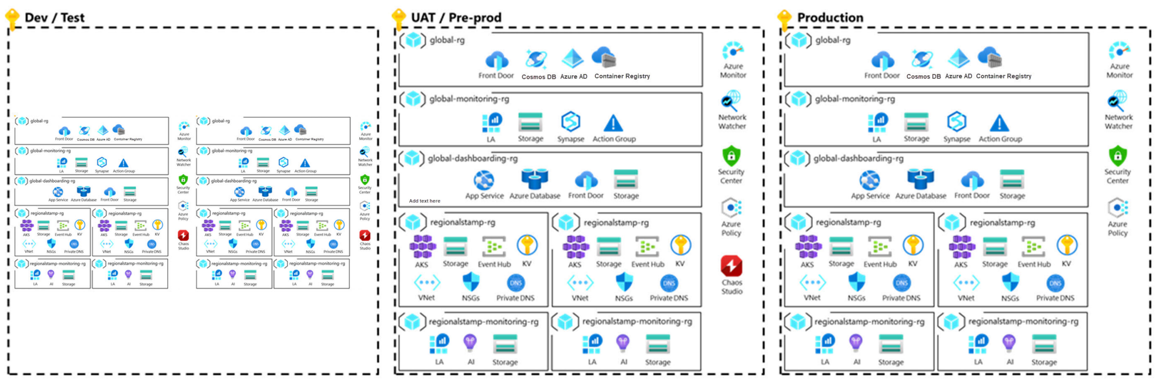


Application environments

View the following video for an overview of recommendations for application environments.

https://learn-video.azurefd.net/vod/player?id=7e6e6390-9f32-4c9e-88da-497a604db319&locale=en-us&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-deployment-testing

You need various types of environments to validate and stage deployment operations. The types have different capabilities and lifecycles. Some environments might reflect the production environment and be long lived, and others might be short lived and have fewer capabilities than production. Setting up these environments early in the development cycle helps to ensure agility, separation of production and preproduction assets, and thorough testing of operations before release to the production environment. All environments should reflect the production environment as much as possible, although you can apply simplifications to lower environments as needed. This diagram shows a mission-critical architecture:



multiple - E2E environments *Pre-production environment* *Production environment*

There are some common considerations:

- Components shouldn't be shared across environments. Possible exceptions are downstream security appliances like firewalls and source locations for synthetic test data.
- All environments should use infrastructure as code (IaC) artifacts like Terraform or Azure Resource Manager (ARM) templates.

Development environments

View the following video for information about ephemeral development environments and automated feature validation.

https://www.microsoft.com/en-us/videoplayer/embed/RE50Gm9?postJslIMsg=true

Design considerations

- **Capabilities**. Lower requirements for reliability, capacity, and security are acceptable for development environments.
- **Lifecycle**. Development environments should be created as required and exist for a short time. Shorter lifecycles help prevent configuration drift from the code base and reduce costs. Also, development environments often share the lifecycle of a feature branch.
- **Density**. Teams need multiple environments to support parallel feature development. They can coexist within a single subscription.

Design recommendations

- Keep the environment in a dedicated subscription with context set for development purposes.
- Use an automated process to deploy code from a feature branch to a development environment.

Test or staging environments

These environments are used for testing and validation. Many test cycles are performed to ensure bug-free deployment to production. Appropriate tests for a mission-critical workload are described in the Continuous validation and testing section.

Design considerations

- **Capabilities**. These environments should reflect the production environment for reliability, capacity, and security. In the absence of a production load, use a synthetic user load to provide realistic metrics and valuable health modeling input.
- **Lifecycle**. These environments are short lived. They should be destroyed after test validations are complete.

- **Density**. You can run many independent environments in one subscription. You should also consider using multiple environments, each in a dedicated subscription.

> ⓘ **Note**
>
> Mission-critical applications should be subjected to rigorous testing.
>
> You can perform different test functions in a single environment, and in some cases you'll need to. For example, for chaos testing to provide meaningful results, you must first place the application under load so you can understand how the application responds to injected faults. That's why chaos testing and load testing are typically performed in parallel.

Design recommendations

- Ensure that at that least one staging environment fully reflects production to enable production-like testing and validation. Capacity within this environment can flex based on the execution of test activities.

- Generate synthetic user load to provide a realistic test case for changes on one of the environments.

  > ⓘ **Note**
  >
  > The **Mission Critical Online** reference implementation provides an example of a **user load generator**.

- Define the number of staging environments and their purposes within the development and release cycle.

Production environments

Design considerations

- **Capabilities**. The highest levels of reliability, capacity, and security functionality for the application are required.

- **Lifecycle**. While the lifecycle of the workload and the infrastructure remains the same, all data, including monitoring and logging, need special management. For example, management is required for backup and recovery.

- **Density**. For some applications, you might want to consider using different production environments to cater to different clients, users, or business functionalities.

Design recommendations

Have a clear governance boundary for production and lower environments. Place each environment type in a dedicated subscription to achieve that goal. This segmentation ensures that resource utilization in lower environments doesn't affect production quotas. Dedicated subscriptions also set access boundaries.

Ephemeral blue/green deployments

A blue/green deployment model requires at least two identical deployments. The blue deployment is the active one that serves user traffic in production. The green deployment is the new one that's prepared and tested to receive traffic. After the green deployment is completed and tested, traffic is gradually directed from blue to green. If the load transfer is successful, the green deployment becomes the new active deployment. The old blue deployment can then be decommissioned via a phased process. However, if there are problems in the new deployment, it can be aborted, and traffic can either remain in the old blue deployment or be redirected to it.

Azure Mission-Critical recommends a blue/green deployment approach where infrastructure *and applications* are deployed together as part of a deployment stamp. So rolling out a change to the infrastructure or application always results in a green deployment that contains both layers. This approach enables you to fully test and validate the effect of the change against the infrastructure and application end-to-end before you redirect user traffic to it. The approach increases confidence in releasing changes and enables zero-downtime upgrades because compatibilities with downstream dependencies like the Azure platform, resource providers, and IaC modules can be validated.

Design considerations

- **Technology capabilities**. Take advantage of the built-in deployment features in Azure services. For example, Azure App Service provides secondary deployment slots that can be swapped after a deployment. With Azure Kubernetes Service (AKS), you can use a separate pod deployment on each node and update the service definition.

- **Deployment duration**. The deployment might take longer to complete because the stamp contains the infrastructure and application rather than just the changed component. This, however, is acceptable because the risk of all components not working as expected overrides the time concerns.

- **Cost impact**. There's an additional cost because of the two side-by-side deployments, which must coexist until the deployment is complete.

- **Traffic transition**. After the new deployment is validated, traffic must be transitioned from the blue environment to the green one. This transition requires orchestration of user traffic between the environments. The transition should be fully automated.

- **Lifecycle**. Mission-critical deployment stamps should be considered ephemeral. Using short-lived stamps creates a fresh start each time, before resources are provisioned.

Design recommendations

- Adopt a blue/green deployment approach to release all production changes. Deploy all infrastructure and the application each time, for any type of change, to achieve a consistent state and zero downtime. Although you can reuse environments, we don't recommend it for mission-critical workloads. Treat each regional deployment stamp as ephemeral with a lifecycle that's tied to that of a single release.

- Use a global load balancer, like Azure Front Door, to orchestrate the automated transition of user traffic between the blue and green environments.

- To transition traffic, add a green back-end endpoint that uses a low traffic to volume weight, like 10 percent. After you verify that the low traffic volume on the green deployment works and provides the expected application health, gradually increase traffic. While doing so, apply a short ramp-up period to catch faults that might not immediately be apparent.

  After all traffic is transitioned, remove the blue back end from existing connections. For instance, remove the back end from the global load balancer service, drain queues, and detach other associations. Doing so helps to optimize the cost of maintaining secondary production infrastructure and ensure that new environments are free of configuration drift.

  At this point, decommission the old and now inactive blue environment. For the next deployment, repeat the process with blue and green reversed.

Subscription-scoped deployment

Depending on the scale requirements of your application, you might need multiple production subscriptions to serve as scale units.

View the following video to get an overview of recommendations for scoping subscriptions for a mission-critical application.

https://learn-video.azurefd.net/vod/player?id=013a2a82-dc85-4282-98ed-b1afe50afd41&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-deployment-testing&locale=en-us

Design considerations

- **Scalability**. For high-scale application scenarios with significant volumes of traffic, design the solution to scale across multiple Azure subscriptions so that the scale limits of a single subscription don't constrain scalability.

  > **Important**
  >
  > The use of multiple subscriptions necessitates additional CI/CD complexity, which must be appropriately managed. Therefore, we recommend multiple subscriptions only in extreme scale scenarios, where the limits of a single subscription are likely to become a limitation.

- **Environment boundaries**. Deploy production, development, and test environments into separate subscriptions. This practice ensures that lower environments don't contribute toward scale limits. It also reduces the risk of lower-environment updates polluting production by providing a clear management and identity boundary.

- **Governance**. When you need multiple production subscriptions, consider using a dedicated application management group to simplify policy assignment via a policy aggregation boundary.

Design recommendations

- Deploy each regional deployment stamp in a dedicated subscription to ensure that the subscription limits apply only within the context of a single deployment stamp and not across the application as a whole. Where appropriate, you might consider using multiple deployment stamps within a single region, but you should deploy them across independent subscriptions.

- Place global shared resources in a dedicated subscription to enable consistent regional subscription deployment. Avoid using a specialized deployment for the primary region.

Continuous validation and testing

Testing is a critical activity that allows you to fully validate the health of your application code and infrastructure. More specifically, testing allows you to meet your standards for reliability, performance, availability, security, quality, and scale. Testing must be well defined and applied as part of your application design and DevOps strategy. Testing is a key concern during the local developer process (the inner loop) and as a part of the complete DevOps lifecycle (the outer loop), which is when code starts on the path from release pipeline processes toward the production environment.

View the following video to get an overview of continuous validation and testing.

https://learn-video.azurefd.net/vod/player?id=fc7842c3-7c7a-44dc-ad87-838aa51d0000&locale=en-us&embedUrl=%2Fazure%2Fwell-architected%2Fmission-critical%2Fmission-critical-deployment-testing

This section focuses on outer loop testing. It describes various types of tests.

Test	Description
Unit testing	Confirms that application business logic works as expected. Validates the overall effect of code changes.
Smoke testing	Identifies whether infrastructure and application components are available and function as expected. Typically, only a single virtual user session is tested. The outcome should be that the system responds with expected values and behavior. Common smoke testing scenarios include reaching the HTTPS endpoint of a web application, querying a database, and simulating a user flow in the application.
UI testing	Validates that application user interfaces are deployed and that user interface interactions function as expected. You should use UI automation tools to drive automation. During a UI test, a script should mimic a realistic user scenario and complete a series of steps to execute actions and achieve an intended outcome.

Test	Description
Load testing	Validates scalability and application operation by increasing load rapidly and/or gradually until a predetermined threshold is reached. Load tests are typically designed around a particular user flow to verify that application requirements are satisfied under a defined load.
Stress testing	Applies activities that overload existing resources to determine solution limits and verify the system's ability to recover gracefully. The main goal is to identify potential performance bottlenecks and scale limits. Conversely, scale down the computing resources of the system and monitor how it behaves under load and determine whether it can recover.
Performance testing	Combines aspects of load and stress testing to validate performance under load and establish benchmark behaviors for application operation.
Chaos testing	Injects artificial failures into the system to evaluate how it reacts and to validate the effectiveness of resiliency measures, operational procedures, and mitigations. Shutting down infrastructure components, purposely degrading performance, and introducing application faults are examples of tests that can be used to verify that the application will react as expected when the scenarios actually occur.
Penetration testing	Ensures that an application and its environment meet the requirements of an expected security posture. The goal is to identify security vulnerabilities. Security testing can include end-to-end software supply chain and package dependencies, with scanning and monitoring for known Common Vulnerabilities and Exposures (CVE).

Design considerations

- **Automation**. Automated testing is essential to validate application or infrastructure changes in a timely and repeatable manner.

- **Test order**. The order in which tests are conducted is a critical consideration because of various dependencies, like the need to have a running application environment. Test duration also influences order. Tests with shorter running times should run earlier in the cycle when possible to increase testing efficiency.

- **Scalability limits**. Azure services have different soft and hard limits. Consider using load testing to determine whether a system risks exceeding them during the expected production load. Load testing can also be useful for setting appropriate thresholds for autoscaling. For services that don't support autoscaling, load testing can help you fine-tune automated operational procedures.

  Inability of system components, like active/passive network components or databases, to appropriately scale can be restrictive. Stress testing can help identify limitations.

- **Failure mode analysis**. Introducing faults into the application and underlying infrastructure and evaluating the effect is essential to assessing the solution's redundancy mechanisms. During this analysis, identify the risk, impact, and breadth of impact (partial or full outage). For an example analysis that was created for the Mission Critical Online reference implementation, see Outage risks of individual components.

- **Monitoring**. You should capture and analyze test results individually and also aggregate them to assess trends over time. You should continually evaluate test results for accuracy and coverage.

Design recommendations

View the following video to see how resiliency testing can be integrated with Azure DevOps CI/CD pipelines.

https://www.microsoft.com/en-us/videoplayer/embed/RE4Y50k?postJslIMsg=true

- Ensure consistency by automating all testing on infrastructure and application components. Integrate the tests in CI/CD pipelines to orchestrate and run them where applicable. For information about technology options, see DevOps tools.

- Treat all test artifacts as code. They should be maintained and version controlled along with other application code artifacts.

- Align the SLA of the test infrastructure with the SLA for deployment and testing cycles.

- Run smoke tests as part of every deployment. Also run extensive load tests along with stress and chaos tests to validate that application performance and operability is maintained.
  - Use load profiles that reflect real peak usage patterns.
  - Run chaos experiments and failure injection tests at the same time as load tests.

> **Tip**
>
> **Azure Chaos Studio** is a native suite of chaos experimentation tools. The tools make it easy to conduct chaos experiments and inject faults within Azure services and application components.

Chaos Studio provides built-in chaos experiments for common fault scenarios and supports custom experiments that target infrastructure and application components.

- If database interactions, like the creation of records, are required for load or smoke tests, use test accounts that have reduced privileges and make test data separable from real user content.

- Scan and monitor the end-to-end software supply chain and package dependencies for known CVEs.
  - Use Dependabot for GitHub repositories to ensure the repository is automatically up to date with the latest releases of packages and applications that it depends on.

Infrastructure as code deployments

Infrastructure as code (IaC) treats infrastructure definitions as source code that's version controlled along with other application artifacts. Using IaC promotes code consistency across environments, eliminates the risk of human error during automated deployments, and provides traceability and rollback. For blue/green deployments, the use of IaC with fully automated deployments is imperative.

A mission-critical IaC repository has two distinct definitions that are mapped to global and regional resources. For information about these types of resources, see the core architecture pattern.

Design considerations

- **Repeatable infrastructure**. Deploy mission-critical workloads in a way that generates the same environment every time. IaC should be the primary model.

- **Automation**. All deployments must be fully automated. Human processes are error prone.

Design recommendations

- Apply IaC, ensuring that all Azure resources are defined in declarative templates and maintained in a source control repository. Templates are deployed and resources are provisioned automatically from source control via CI/CD pipelines. We don't recommend the use of imperative scripts.

- Prohibit manual operations against all environments. The only exception is fully independent developer environments.

DevOps tools

The effective use of deployment tools is critical to overall reliability because DevOps processes affect the overall function and application design. For example, failover and scale operations might depend on automation that's provided by DevOps tools. Engineering teams must understand the effect of the unavailability of a deployment service with respect to the overall workload. Deployment tooling must be reliable and highly available.

Microsoft provides two Azure-native toolsets, GitHub Actions and Azure Pipelines, that can effectively deploy and manage a mission-critical application.

Design considerations

- **Technology capabilities**. The capabilities of GitHub and Azure DevOps overlap. You can use them together to get the best of both. A common approach is to store code repositories in GitHub.com or GitHub AE while using Azure Pipelines for deployment.

  Be aware of the complexity that's added when you use multiple technologies. Evaluate a rich feature set against overall reliability.

- **Regional availability**. In terms of maximum reliability, the dependency on a single Azure region represents an operational risk.

  For example, say traffic is spread over two regions: Region 1 and Region 2. Region 2 hosts the Azure DevOps tooling instance. Suppose there's an outage in Region 2 and the instance isn't available. Region 1 automatically handles all traffic and needs to deploy extra scale units to provide a good failover experience. But it won't be able to because of the Azure DevOps dependency in Region 2.

- **Data replication**. Data, including metadata, pipelines, and source code, should be replicated across regions.

Design recommendations

- Both technologies are hosted in a single Azure region, which might make your disaster recovery strategy restrictive.

GitHub Actions is well-suited for build tasks (continuous integration) but might lack features for complex deployment tasks (continuous deployment). Given the rich feature set of Azure DevOps, we recommend it for mission-critical deployments. However, you should make a choice after you assess trade-offs.

- Define an availability SLA for deployment tooling and ensure alignment with broader application reliability requirements.

- For multi-region scenarios that use an active/passive or active/active deployment configuration, make sure that failover orchestration and scaling operations can continue to function even if the primary region hosting deployment toolsets becomes unavailable.

Branching strategy

There are many valid approaches to branching. You should choose a strategy that ensures maximum reliability. A good strategy enables parallel development, provides a clear path from development to production, and supports fast releases.

Design considerations

- **Minimize access**. Developers should do their work in *feature/** and *fix/** branches. These branches become entry points for changes. Role-based restrictions should be applied to branches as part of the branching strategy. For example, only allow administrators to create release branches or enforce naming conventions for branches.

- **Accelerated process for emergencies**. The branching strategy should allow hotfixes to be merged into *main* as soon as practical. That way, future releases contain the fix, and recurrence of the problem is avoided. Use this process only for minor changes that address urgent problems, and use it with restraint.

Design recommendations

- Prioritize the use of GitHub for source control.

> **Important**
>
> Create a branching strategy that details *feature* work and *releases* as a minimum, and use branch policies and permissions to ensure that the strategy is appropriately enforced.

- Trigger an automated testing process to validate code change contributions before they're accepted. Team members must also review changes.

- Treat the *main* branch as a continuously forward-moving and stable branch that's primarily used for integration testing.
  - Ensure that changes are made to *main* only via PRs. Use a branch policy to prohibit direct commits.
  - Every time a PR is merged into *main*, it should automatically kick off a deployment against an integration environment.
  - The *main* branch should be considered stable. It should be safe to create a release from *main* at any given time.

- Use dedicated *release/** branches that are created from the *main* branch and used to deploy to production environments. *release/** branches should remain in the repository and can be used to patch a release.

- Document a hotfix process and use it only when needed. Create hotfixes in a *fix/** branch for subsequent merging into the release branch and deployment to production.

AI for DevOps

You can apply AIOps methodologies in CI/CD pipelines to supplement traditional testing approaches. Doing so enables detection of potential regressions or degradations and allows deployments to be preemptively stopped to prevent potential negative impacts.

Design considerations

- **Volume of telemetry data**. CI/CD pipelines and DevOps processes emit a wide variety of telemetry for machine learning models. The telemetry ranges from test results and deployment outcomes to operational data about test components from composite deployment stages.

- **Scalability**. Traditional data processing approaches like Extract, Transform, Load (ETL) might not be able to scale throughput to keep up with the growth of deployment telemetry and application observability data. You can use modern analytics approaches that don't require ETL and data movement, like data virtualization, to enable ongoing analysis by AIOps models.

- **Deployment changes**. Changes in deployment need to be stored for automated analysis and correlation to deployment outcomes.

Design recommendations

- Define the DevOps process data to collect and how to analyze it. Telemetry, like test execution metrics and time series data of changes within each deployment, is important.
  - Expose application observability data from staging, test, and production environments for analysis and correlation within AIOps models.
- Adopt the MLOps workflow.
- Develop analytical models that are context-aware and dependency-aware to provide predictions with automated feature engineering to address schema and behavior changes.
- Operationalize models by registering and deploying the best-trained models within deployment pipelines.

Next step

Review the security considerations.

Security

Security considerations for mission-critical workloads on Azure

Article • 03/15/2023

Security is a one of the foundational design principles and also a key design area that must be treated as a first-class concern within the mission-critical architectural process.

Given that the primary focus of a mission-critical design is to maximize reliability so that the application remains performant and available, the security considerations and recommendations applied within this design area will focus on mitigating threats with the capacity to impact availability and hinder overall reliability. For example, successful Denial-Of-Service (DDoS) attacks are known to have a catastrophic impact on availability and performance. How an application mitigates those attack vectors, such as SlowLoris will impact the overall reliability. So, the application must be fully protected against threats intended to directly or indirectly compromise application reliability to be truly mission critical in nature.

It's also important to note that there are often significant trade-offs associated with a hardened security posture, particularly with respect to performance, operational agility, and in some cases reliability. For example, the inclusion of inline Network Virtual Appliances (NVA) for Next-Generation Firewall (NGFW) capabilities, such as deep packet inspection, will introduce a significant performance penalty, additional operational complexity, and a reliability risk if scalability and recovery operations are not closely aligned with that of the application. It's therefore essential that additional security components and practices intended to mitigate key threat vectors are also designed to support the reliability target of an application, which will form a key aspect of the recommendations and considerations presented within this section.

> **Important**
>
> This article is part of the **Azure Well-Architected mission-critical workload** series. If you aren't familiar with this series, we recommend you start with **what is a mission-critical workload?**
>
> **Mission-Critical open source project**
>
> The **reference implementations** are part of an open source project available on GitHub. The code assets adopt a Zero Trust model to structure and guide the security design and implementation approach.

Alignment with the Zero Trust model

The Microsoft Zero Trust model provides a proactive and integrated approach to applying security across all layers of an application. The guiding principles of Zero Trust strives to explicitly and continuously verify every transaction, assert least privilege, use intelligence, and advanced detection to respond to threats in near real-time. It's ultimately centered on eliminating trust inside and outside of application perimeters, enforcing verification for anything attempting to connect to the system.

Design considerations

As you assess the security posture of the application, start with these questions as the basis for each consideration.

- Continuous security testing to validate mitigations for key security vulnerabilities.
  - *Is security testing performed as a part of automated CI/CD processes?*
  - *If not, how often is specific security testing performed?*
  - *Are test outcomes measured against a desired security posture and threat model?*

- Security level across all lower-environments.
  - *Do all environments within the development lifecycle have the same security posture as the production environment?*

- Authentication and Authorization continuity in the event of a failure.
  - *If authentication or authorization services are temporarily unavailable, will the application be able to continue to operate?*

- Automated security compliance and remediation.
  - *Can changes to key security settings be detected*?
  - *Are responses to remediate non-compliant changes automated?*

- Secret scanning to detect secrets before code is committed to prevent any secret leaks through source code repositories.
  - *Is authentication to services possible without having credentials as a part of code?*

- Secure the software supply chain.
  - *Is it possible to track Common Vulnerabilities and Exposures (CVEs) within utilized package dependencies?*
  - *Is there an automated process for updating package dependencies?*

- Data protection key lifecycles.
  - *Can service-managed keys be used for data integrity protection?*

- *If customer-managed keys are required, how is the secure and reliable key lifecycle?*
- CI/CD tooling should require Azure AD service principals with sufficient subscription level access to facilitate control plane access for Azure resource deployments to all considered environment subscriptions.
  - *When application resources are locked down within private networks, is there a private data-plane connectivity path so that CI/CD tooling can perform application level deployments and maintenance.*
    - This introduces additional complexity and requires a sequence within the deployment process through requisite private build agents.

Design recommendations

- Use Azure Policy to enforce security and reliability configurations for all service, ensuring that any deviation is either remediated or prohibited by the control plane at configuration-time, helping to mitigate threats associated with 'malicious admin' scenarios.
- Use Azure AD Privileged Identity Management (PIM) within production subscriptions to revoke sustained control plane access to production environments. This will significantly reduce the risk posed from 'malicious admin' scenarios through additional 'checks and balances'.
- Use Azure Managed Identities for all services that support the capability, since it facilitates the removal of credentials from application code and removes the operational burden of identity management for service to service communication.
- Use Azure AD Role Based Access Control (RBAC) for data plane authorization with all services that support the capability.
- Use first-party Microsoft identity platform authentication libraries within application code to integrate with Azure AD.
- Consider secure token caching to allow for a degraded but available experience if the chosen identity platform, isn't available or is only partially available for application authorization.
  - If the provider is unable to issue new access tokens, but still validates existing ones, the application and dependent services can operate without issues until their tokens expire.
  - Token caching is typically handled automatically by authentication libraries (such as MSAL).

- Use Infrastructure-as-Code (IaC) and automated CI/CD pipelines to drive updates to all application components, including under failure circumstances.
  - Ensure CI/CD tooling service connections are safeguarded as critical sensitive information, and shouldn't be directly available to any service team.
  - Apply granular RBAC to production CD pipelines to mitigate 'malicious admin' risks.
  - Consider the use of manual approval gates within production deployment pipelines to further mitigate 'malicious admin' risks and provide additional technical assurance for all production changes.
    - Additional security gates may come at a trade-off in terms of agility and should be carefully evaluated, with consideration given to how agility can be maintained even with manual gates.

- Define an appropriate security posture for all lower environments to ensure key vulnerabilities are mitigated.
  - Do not apply the same security posture as production, particularly with regard to data exfiltration, unless regulatory requirements stipulate the need to do so, since this will significantly compromise developer agility.

- Enable Microsoft Defender for Cloud (formerly known as Azure Security Center) for all subscriptions that contain the resources for a mission-critical workload.
  - Use Azure Policy to enforce compliance.
  - Enable Azure Defender for all services that support the capability.

- Embrace DevSecOps and implement security testing within CI/CD pipelines.
  - Test results should be measured against a compliant security posture to inform release approvals, be they automated or manual.
  - Apply security testing as part of the CD production process for each release.
    - If security testing each release jeopardizes operational agility, ensure a suitable security testing cadence is applied.

- Enable secret scanning and dependency scanning within the source code repository.

Threat modeling

Threat modeling provides a risk based approach to security design, using identified potential threats to develop appropriate security mitigations. There are many possible threats with varying probabilities of occurrence, and in many cases threats can chain in unexpected, unpredictable, and even chaotic ways. This complexity and uncertainty is why traditional technology requirement based security approaches are largely unsuitable for mission-critical cloud applications. Expect the process of threat modeling for a mission-critical application to be complex and unyielding.

To help navigate these challenges, a layered defense-in-depth approach should be applied to define and implement compensating mitigations for modeled threats, considering the following defensive layers.

1. The Azure platform with foundational security capabilities and controls.
2. The application architecture and security design.
3. Security features built-in, enabled, and deployable applied to secure Azure resources.
4. Application code and security logic.
5. Operational processes and DevSecOps.

> **Note**
>
> When deploying within an Azure landing zone, be aware that an additional threat mitigation layer through the provisioning of centralized security capabilities is provided by the landing zone implementation.

Design considerations

STRIDE provides a lightweight risk framework for evaluating security threats across key threat vectors.

- Spoofed Identity: Impersonation of individuals with authority. For example, an attacker impersonating another user by using their -
  - Identity
  - Authentication
- Tampering Input: Modification of input sent to the application, or the breach of trust boundaries to modify application code. For example, an attacker using SQL Injection to delete data in a database table.
  - Data integrity
  - Validation
  - Blocklisting/allowlisting
- Repudiation of Action: Ability to refute actions already taken, and the ability of the application to gather evidence and drive accountability. For example, the deletion of critical data without the ability to trace to a malicious admin.
  - Audit/logging
  - Signing

- Information Disclosure: Gaining access to restricted information. An example would be an attacker gaining access to a restricted file.
  - Encryption
  - Data exfiltration
  - Man-in-the-middle attacks
- Denial of Service: Malicious application disruption to degrade user experience. For example, a DDoS botnet attack such as Slowloris.
  - DDoS
  - Botnets
  - CDN and WAF capabilities
- Elevation of Privilege: Gaining privileged application access through authorization exploits. For example, an attacker manipulating a URL string to gain access to sensitive information.
  - Remote code execution
  - Authorization
  - Isolation

Design recommendations

- Allocate engineering budget within each sprint to evaluate potential new threats and implement mitigations.

- Conscious effort should be applied to ensure security mitigations are captured within a common engineering criteria to drive consistency across all application service teams.

- Start with a service by service level threat modeling and unify the model by consolidating the thread model on application level.

Network intrusion protection

Preventing unauthorized access to a mission-critical application and encompassed data is vital to maintain availability and safeguard data integrity.

Design considerations

- Zero Trust assumes a breached state and verifies each request as though it originates from an uncontrolled network.
  - An advanced zero-trust network implementation employs micro-segmentation and distributed ingress/egress micro-perimeters.

- Azure PaaS services are typically accessed over public endpoints. Azure provides capabilities to secure public endpoints or even make them entirely private.
  - Azure Private Link/Private Endpoints provide dedicated access to an Azure PaaS resource using private IP addresses and private network connectivity.
  - Virtual Network Service Endpoints provide service-level access from selected subnets to selected PaaS services.
  - Virtual Network Injection provides dedicated private deployments for supported services, such as App Service through an App Service Environment.
    - Management plane traffic still flows through public IP addresses.

- For supported services, Azure Private Link using Azure Private Endpoints addresses data exfiltration risks associated with Service Endpoints, such as a malicious admin writing data to an external resource.

- When restricting network access to Azure PaaS services using Private Endpoints or Service Endpoints, a secure network channel will be required for deployment pipelines to access both the Azure control plane and data plane of Azure resources in order to deploy and manage the application.
  - Private self-hosted build agents deployed onto a private network as the Azure resource can be used as a proxy to execute CI/CD functions over a private connection. A separate virtual network should be used for build agents.
    - Connectivity to the private build agents from CI/CD tooling is required.
  - An alternative approach is to modify the firewall rules for the resource on-the-fly within the pipeline to allow a connection from an Azure DevOps agent public IP address, with the firewall subsequently removed after the task is completed.
    - However, this approach is only applicable for a subset of Azure services. For example, this isn't feasible for private AKS clusters.
  - To perform developer and administrative tasks on the application service jump boxes can be used.

- The completion of administration and maintenance tasks is a further scenario requiring connectivity to the data plane of Azure resources.

- Service Connections with a corresponding Azure AD service principal can be used within Azure DevOps to apply RBAC through Azure AD.

- Service Tags can be applied to Network Security Groups to facilitate connectivity with Azure PaaS services.

- Application Security Groups don't span across multiple virtual networks.

- Packet capture in Azure Network Watcher is limited to a maximum period of five hours.

Design recommendations

- Limit public network access to the absolute minimum required for the application to fulfill its business purpose to reduce the external attack surface.
  - Use Azure Private Link to establish private endpoints for Azure resources that require secure network integration.
  - Use hosted private build agents for CI/CD tooling to deploy and configure Azure resources protected by Azure Private Link.
    - Microsoft-hosted agents won't be able to directly connect to network integrated resources.

- When dealing with private build agents, never open an RDP or SSH port directly to the internet.
  - Use Azure Bastion to provide secure access to Azure Virtual Machines and to perform administrative tasks on Azure PaaS over the Internet.

- Use a DDoS standard protection plan to secure all public IP addresses within the application.

- Use Azure Front Door with web application firewall policies to deliver and help protect global HTTP/S applications that span multiple Azure regions.
  - Use Header ID validation to lock down public application endpoints so they only accept traffic originating from the Azure Front Door instance.

- If additional in-line network security requirements, such as deep packet inspection or TLS inspection, mandate the use of Azure Firewall Premium or Network Virtual Appliance (NVA), ensure it's configured for maximum high availability and redundancy.

- If packet capture requirements exist, use Network Watcher packets to capture despite the limited capture window.

- Use Network Security Groups and Application Security Groups to micro-segment application traffic.
  - Avoid using a security appliance to filter intra-application traffic flows.
  - Consider the use of Azure Policy to enforce specific NSG rules are always associated with application subnets.

- Enable NSG flow logs and feed them into Traffic Analytics to gain insights into internal and external traffic flows.

- Use Azure Private Link/Private Endpoints, where available, to secure access to Azure PaaS services within the application design. For information on Azure services that support Private Link, see Azure Private Link availability.

- If Private Endpoint isn't available and data exfiltration risks are acceptable, use Virtual Network Service Endpoints to secure access to Azure PaaS services from within a virtual network.
  - Don't enable virtual network service endpoints by default on all subnets as this will introduce significant data exfiltration channels.

- For hybrid application scenarios, access Azure PaaS services from on-premises via ExpressRoute with private peering.

> **Note**
>
> When deploying within an Azure landing zone, be aware that network connectivity to on-premises data centers is provided by the landing zone implementation. One approach is by using ExpressRoute configured with private peering.

Data integrity protection

Encryption is a vital step toward ensuring data integrity and is ultimately one of the most important security capabilities that can be applied to mitigate a wide array of threats. This section will therefore provide key considerations and recommendations related to encryption and key management in order to safeguard data without compromising application reliability.

Design considerations

- Azure Key Vault has transaction limits for keys and secrets, with throttling applied per vault within a certain period.

- Azure Key Vault provides a security boundary since access permissions for keys, secrets, and certificates are applied at a vault level.
  - Key Vault access policy assignments grant permissions separately to keys, secrets, or certificates.
    - Granular object-level permissions to a specific key, secret, or certificate are now possible.

- After a role assignment is changed, there's a latency of up to 10 minutes (600 seconds) for the role to be applied.
  - There's an Azure AD limit of 2,000 Azure role assignments per subscription.

- Azure Key Vault underlying hardware security modules (HSMs) are FIPS 140-2 Level 2 compliant.

    - A dedicated Azure Key Vault managed HSM is available for scenarios requiring FIPS 140-2 Level 3 compliance.

- Azure Key Vault provides high availability and redundancy to help maintain availability and prevent data loss.

- During a region failover, it may take a few minutes for the Key Vault service to fail over.
    - During a failover Key Vault will be in a read-only mode, so it won't be possible to change key vault properties such as firewall configurations and settings.

- If private link is used to connect to Azure Key Vault, it may take up to 20 minutes for the connection to be re-established during a regional failover.

- A backup creates a point-in-time snapshot of a secret, key, or certificate, as an encrypted blob that can't be decrypted outside of Azure. To get usable data from the blob, it must be restored into a Key Vault within the same Azure subscription and Azure geography.
    - Secrets may renew during a backup, causing a mismatch.

- With service-managed keys, Azure will perform key management functions, such as rotation, thereby reducing the scope of application operations.

- Regulatory controls may stipulate the use of customer-managed keys for service encryption functionality.

- When traffic moves between Azure data centers, MACsec data-link layer encryption is used on the underlying network hardware to secure data in-transit outside of the physical boundaries not controlled by Microsoft or on behalf of Microsoft.

Design recommendations

- Use service-managed keys for data protection where possible, removing the need to manage encryption keys and handle operational tasks such as key rotation.
    - Only use customer-managed keys when there's a clear regulatory requirement to do so.

- Use Azure Key Vault as a secure repository for all secrets, certificates, and keys if additional encryption mechanisms or customer-managed keys need considered.
    - Provision Azure Key Vault with the soft delete and purge policies enabled to allow retention protection for deleted objects.
    - Use HSM backed Azure Key Vault SKU for application production environments.

- Deploy a separate Azure Key Vault instance within each regional deployment stamp, providing fault isolation and performance benefits through localization, as well as navigating the scale limits imposed by a single Key Vault instance.
  - Use a dedicated Azure Key Vault instance for application global resources.

- Follow a least privilege model by limiting authorization to permanently delete secrets, keys, and certificates to specialized custom Azure AD roles.

- Ensure encryption keys, and certificates stored within Key Vault are backed up, providing an offline copy in the unlikely event Key Vault becomes unavailable.

- Use Key Vault certificates to manage certificate procurement and signing.

- Establish an automated process for key and certificate rotation.
  - Automate the certificate management and renewal process with public certificate authorities to ease administration.
    - Set alerting and notifications, to supplement automated certificate renewals.

- Monitor key, certificate, and secret usage.
  - Define alerts for unexpected usage within Azure Monitor.

Policy-driven governance

Security conventions are ultimately only effective if consistently and holistically enforced across all application services and teams. Azure Policy provides a framework to enforce security and reliability baselines, ensuring continued compliance with a common engineering criteria for a mission-critical application. More specifically, Azure Policy forms a key part of the Azure Resource Manager (ARM) control plane, supplementing RBAC by restricting what actions authorized users can perform, and can be used to enforce vital security and reliability conventions across utilized platform services.

This section will therefore explore key considerations and recommendations surrounding the use of Azure Policy driven governance for a mission-critical application, ensuring security and reliability conventions are continuously enforced.

Design considerations

- Azure Policy provides a mechanism to drive compliance by enforcing security and reliability conventions, such as the use of Private Endpoints or the use of Availability Zones.

ⓘ **Note**

When deploying within an Azure landing zone, be aware that the enforcement of centralized baseline policy assignments will likely be applied in the implementation for landing zone management groups and subscriptions.

- Azure Policy can be used to drive automated management activities, such as provisioning and configuration.
  - Resource Provider registration.
  - Validation and approval of individual Azure resource configurations.
- Azure Policy assignment scope dictates coverage and the location of Azure Policy definitions informs the reusability of custom policies.
- Azure Policy has several limits, such as the number of definitions at any particular scope.
- It can take several minutes for the execution of Deploy If Not Exist (DINE) policies to occur.
- Azure Policy provides a critical input for compliance reporting and security auditing.

Design recommendations

- Map regulatory and compliance requirements to Azure Policy definitions.
  - For example, if there are data residency requirements, a policy should be applied to restrict available deployment regions.
- Define a common engineering criteria to capture secure and reliable configuration definitions for all utilized Azure services, ensuring this criteria is mapped to Azure Policy assignments to enforce compliance.
  - For example, apply an Azure Policy to enforce the use of Availability Zones for all relevant services, ensuring reliable intra-region deployment configurations.

> The Mission Critical reference implementation contain a wide array of security and reliability centric policies to define and enforce a sample common engineering criteria.

- Monitor service configuration drift, relative to the common engineering criteria, using Azure Policy.

> For mission-critical scenarios with multiple production subscriptions under a dedicated management group, prioritize assignments at the management group

> scope.

- Use built-in policies where possible to minimize operational overhead of maintaining custom policy definitions.
- Where custom policy definitions are required, ensure definitions are deployed at suitable management group scope to allow for reuse across encompassed environment subscriptions to this allow for policy reuse across production and lower environments.
  - When aligning the application roadmap with Azure roadmaps, use available Microsoft resources to explore if critical custom definitions could be incorporated as built-in definitions.

> ⓘ **Note**
>
> When deploying within an Azure landing zone, consider deploying custom Azure Policy Definitions within the intermediate company root management group scope to enable reuse across all applications within the broader Azure estate. In a landing zone environment, certain centralized security policies will be applied by default within higher management group scopes to enforce security compliance across the entire Azure estate. For example, Azure policies should be applied to automatically deploy software configurations through VM extensions and enforce a compliant baseline VM configuration.

- Use Azure Policy to enforce a consistent tagging schema across the application.
  - Identify required Azure tags and use the append policy mode to enforce usage.

> If the application is subscribed to Microsoft Mission-Critical Support, ensure that the applied tagging schema provides meaningful context to enrich the support experience with deep application understanding.

- Export Azure AD activity logs to the global Log Analytics Workspace used by the application.
  - Ensure Azure activity logs are archived within the global Storage Account along with operational data for long-term retention.

> In an Azure landing zone, Azure AD activity logs will also be ingested into the centralized platform Log Analytics workspace. It needs to be evaluated in this case if Azure AD are still required in the global Log Analytics workspace.

- Integrate security information and event management with Microsoft Defender for Cloud (formerly known as Azure Security Center).

IaaS specific considerations when using Virtual Machines

In scenarios where the use of IaaS Virtual Machines is required, some specifics have to be taken into consideration.

Design considerations

- Images are not updated automatically once deployed.
- Updates are not installed automatically to running VMs.
- Images and individual VMs are typically not hardened out-of-the-box.

Design recommendations

- Do not allow direct access via the public Internet to Virtual Machines by providing access to SSH, RDP or other protocols. Always use Azure Bastion and jumpboxes with limited access to a small group of users.
- Restrict direct internet connectivity by using Network Security Groups, (Azure) Firewall or Application Gateways (Level 7) to filter and restrict egress traffic.
- For multi-tier applications consider using different subnets and use Network Security Groups to restrict access in between.
- Prioritize the use of Public Key authentication, when possible. Store secrets in a secure place like Azure Key Vault.
- Protect VMs by using authentication and access control.
- Apply the same security practices as described for mission-critical application scenarios.

Follow and apply security practices for mission-critical application scenarios as described above, when applicable, as well as the Security best practices for IaaS workloads in Azure.

Next step

Review the best practices for operational procedures for mission-critical application scenarios.

Operational procedures

Operational procedures for mission-critical workloads on Azure

Article • 02/01/2023

Reliable and effective operations must be based on the principles of *automation wherever possible* and *configuration as code*. This approach requires a substantial engineering investment in DevOps processes. Automated pipelines are used to deploy application and infrastructure code artifacts. The benefits of this approach include consistent and accurate operational outcomes with minimal manual operational procedures.

This design area explores how to implement effective and consistent operational procedures.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected Framework mission-critical workload** series. If you aren't familiar with this series, we recommend that you start with **What is a mission-critical workload?**.

DevOps processes

DevOps combines development and operational processes and teams into a single engineering function. It encompasses the entire application lifecycle and uses automation and DevOps tooling to conduct deployment operations in a fast, efficient, and reliable way. DevOps processes support and sustain continuous integration and continuous delivery (CI/CD) while fostering a culture of continuous improvement.

The DevOps team for a mission-critical application must be responsible for these tasks:

- Creation and management of application and infrastructure resources via CI/CD automation.
- Application monitoring and observability.
- Azure role-based access control (RBAC) and identity management for application components.
- Network management for application components.
- Cost management for application resources.

DevSecOps expands the DevOps model by integrating security monitoring, application audits, and quality assurance with development and operations throughout the application lifecycle. DevOps teams are needed for security-sensitive and highly regulated scenarios to ensure that security is incorporated throughout the development lifecycle rather than at a specific release stage or gate.

Design considerations

- **Release and update process**. Avoid manual processes for any change to application components or underlying infrastructure. Manual processes can lead to inconsistent results.

- **Dependencies on central IT teams**. DevOps processes can be difficult to apply when there are hard dependencies on centralized functions because these dependencies prevent end-to-end operations.

- **Identity and access management**. DevOps teams can consider granular Azure RBAC roles for various technical functions, like AppDataOps for database management. Apply a zero-trust model across DevOps roles.

Design recommendations

- Define configuration settings and updates as code. Apply change management through code to enable consistent release and update processes, including tasks like key or secret rotation and permissions management. Use pipeline-managed update processes, like scheduled pipeline runs, rather than built-in auto-update mechanisms.

- Don't use central processes or provisioning pipelines for the instantiation or management of application resources. Doing so introduces external application dependencies and additional risk vectors, like those associated with noisy neighbor scenarios.

  If you need to use centralized provisioning processes, ensure that the availability requirements of the dependencies are fully aligned with mission-critical requirements. Central teams must provide transparency so that holistic operationalization of the end-to-end application is achieved.

- Dedicate a proportion of engineering capacity during each sprint to driving fundamental platform improvements and bolstering reliability. We recommend that you allocate 20-40 percent of capacity to these improvements.

- Develop common engineering criteria, reference architectures, and libraries that are aligned with core design principles. Enforce a consistent baseline configuration for reliability, security, and operations via a policy-driven approach that uses Azure Policy.

  You can also use the common engineering criteria and associated artifacts, like Azure policies and Terraform resources for common design patterns, across other workloads within your organization's broader application ecosystem.

- Apply a zero-trust model in critical application environments. Use technologies like Azure AD Privileged Identity Management to ensure that operations are consistent and occur only through CI/CD processes or automated operational procedures.

  Team members shouldn't have standing write access to any environment. You might want to make exceptions in development environments to enable easier testing and debugging.

- Define emergency processes for just-in-time access to production environments. Ensure that break glass accounts exist in case of serious problems with the authentication provider.

- Consider using AIOps to continually improve operational procedures and triggers.

Application operations

The application design and platform recommendations influence operational procedures. There are also operational capabilities provided by various Azure services, particularly for high availability and recovery.

Design considerations

- **Built-in operations of Azure services**. Azure services provide built-in (enabled by default) and configurable platform capabilities, like zonal redundancy and geo-replication. An application's reliability depends on these operations. Certain configurable capabilities incur an additional cost, like the multi-write deployment configuration for Azure Cosmos DB. Avoid building custom solutions unless you absolutely need to.

- **Operational access and execution time**. Most required operations are exposed and accessible through the Azure Resource Manager API or the Azure portal. However, certain operations require assistance from support engineers. For example, a restore from a periodic backup of an Azure Cosmos DB database, or the recovery of a deleted resource, can be performed only by Azure support engineers via a support case. This dependency might affect the downtime of the application. For stateless resources, we recommend that you redeploy instead of waiting for support engineers to try to recover deleted resources.

- **Policy enforcement**. Azure Policy provides a framework for enforcing and auditing security and reliability baselines to ensure compliance with common engineering criteria for mission-critical applications. More specifically, Azure Policy forms a key part of the Azure Resource Manager control plane, supplementing RBAC by restricting the actions that authorized users can perform. You can use Azure Policy to enforce vital security and reliability conventions across platform services.

- **Modification and deletion of resources**. You can lock Azure resources to prevent them from being modified or deleted. However, locks introduce management overhead in deployment pipelines. For most resources, we recommend a robust RBAC process with tight restrictions rather than resource locks.

Design recommendations

- Automate failover procedures. For an active/active model, use a health model and automated scale operations to ensure that no failover intervention is required. For an active/passive model, ensure that failover procedures are automated or at least codified within pipelines.

- Prioritize the use of Azure-native autoscaling for services that support it. For services that don't support native autoscaling, use automated operational processes to scale services. Use scale units with multiple services to achieve scalability.

- Use platform-native capabilities for backup and restore, ensuring that they're aligned with your RTO/RPO and data retention requirements. Define a strategy for long-term backup retention as needed.

- Use built-in capabilities for SSL certificate management and renewal, like those provided by Azure Front Door.

- For external teams, establish a recovery process for resources that require assistance. For example, if the data platform is incorrectly modified or deleted, the recovery methods should be well understood, and a recovery process should be in place. Similarly, establish procedures to manage decommissioned container images in the registry.

- Practice recovery operations in advance, on non-production resources and data, as part of standard business continuity preparations.

- Identify critical alerts and define target audiences and systems. Define clear channels to reach appropriate stakeholders. Send only actionable alerts to avoid white noise and prevent operational stakeholders from ignoring alerts and missing important information. Implement continuous improvement to optimize alerting and remove observed white noise.

- Apply policy-driven governance and Azure Policy to ensure the appropriate use of operational capabilities and a reliable configuration baseline across all application services.

- Avoid the use of resource locks on ephemeral regional resources. Instead, rely on the appropriate use of RBAC and CI/CD pipelines to control operational updates. You can apply resource locks to prevent the deletion of long-lived global resources.

Update management

Mission-critical design strongly endorses the principle of ephemeral stateless application resources. If you apply this principle, you can typically perform an update by using a new deployment and standard delivery pipelines.

Design considerations

- **Alignment with Azure roadmaps**. Align your workload with Azure roadmaps so that platform resources and runtime dependencies are updated regularly.

- **Automatic detection of updates**. Set up processes to monitor and automatically detect updates. Use tools like GitHub Dependabot.

- **Testing and validation**. Test and validate new versions of packages, components, and dependencies in a production context before any release. New versions might contain breaking changes.

- **Runtime dependencies**. Treat runtime dependencies like you would any other change to the application. Older versions might introduce security vulnerabilities and might have a negative effect on performance. Monitor the application runtime environment and keep it up to date.

- **Component hygiene and housekeeping**. Decommission or remove resources that aren't used. For example, monitor container registries and delete old image versions that you aren't using.

Design recommendations

- Monitor these resources and keep them up to date:
  - The application hosting platform. For example, you need to update the Kubernetes version in Azure Kubernetes Service (AKS) regularly, especially given that support for older versions isn't sustained. You also need to update components that run on Kubernetes, like cert-manager and the Azure Key Vault CSI, and align them with the Kubernetes version in AKS.
  - External libraries and SDKs (.NET, Java, Python).
  - Terraform providers.

- Avoid manual operational changes to update components. Consider the use of manual changes only in emergency situations. Ensure that you have a process for reconciling any manual changes back into the source repository to avoid drift and issue recurrence.

- Establish an automated procedure for removing old versions of images from Azure Container Registry.

Secret management

Key, secret, and certificate expirations are a common cause of application outage. Secret management for a mission-critical application must provide the needed security and offer an appropriate level of availability to align with your maximum-reliability requirements. You need to perform key, secret, and certificate rotation on a regular basis by using a managed service or as part of update management, and apply processes for code and configuration changes.

Many Azure services support Azure Active Directory (Azure AD) authentication instead of relying on connection strings / keys. Using Azure AD greatly reduces operational overhead. If you do use a secret management solution, it should integrate with other services, support zonal and regional redundancy, and provide integration with Azure AD for authentication and authorization. Key Vault provides these features.

Design considerations

There are three common approaches to secret management. Each approach reads secrets from the secret store and injects them into the application at a different time.

- **Deployment-time retrieval**. The advantage to this approach is that the secret management solution needs to be available only at deployment time because there aren't direct dependencies after that time. Examples include injecting secrets as environment variables into a Kubernetes deployment or into a Kubernetes secret.

  Only the deployment service principal needs to be able to access secrets, which simplifies RBAC permissions within the secret management system.

  There are, however, disadvantages to this approach. It introduces RBAC complexity in DevOps tooling with regard to controlling service principal access and in the application with regard to protecting retrieved secrets. Also, the security benefits of the secret management solution aren't applied because this approach relies only on access control in the application platform.

  To implement secret updates or rotation, you need to perform a full redeployment.

- **Application-startup retrieval**. In this approach, secrets are retrieved and injected at application startup. The benefit is that you can easily update or rotate secrets. You don't need to store secrets on the application platform. A restart of the application is required to fetch the latest value.

  Common storage choices include Azure Key Vault Provider for Secrets Store CSI Driver and akv2k8s. A native Azure solution, Key Vault referenced app settings, is also available.

  A disadvantage of this approach is that it creates a runtime dependency on the secret management solution. If the secret management solution experiences an outage, application components already running *might* be able to continue serving requests. Any restart or scale-out operation would likely result in failure.

- **Runtime retrieval**. Retrieving secrets at runtime from within the application itself is the most secure approach because even the application platform never has access to secrets. The application needs to authenticate itself to the secret management system.

  However, for this approach, application components require a direct dependency and a connection to the secret management system. This makes it harder to test components individually and usually necessitates the use of an SDK.

Design recommendations

- When possible, use Azure AD authentication to connect to services instead of using connection strings or keys. Use this authentication method together with Azure managed identities so you don't need to store secrets on the application platform.
- Take advantage of the expiry setting in Key Vault, and configure alerting for upcoming expirations. Perform all key, secret, and certificate updates by using the standard release process.
- Deploy Key Vault instances as part of a regional stamp to mitigate the potential effect of a failure to a single deployment stamp. Use a separate instance for global resources. For information about those resources, see the typical architecture pattern for mission-critical workloads.
- To avoid the need to manage service principal credentials or API keys, use managed identities instead of service principals to access Key Vault whenever possible.
- Implement coding patterns to ensure that secrets are re-retrieved when an authorization failure occurs at runtime.
- Apply a fully automated key-rotation process that runs periodically within the solution.
- Use the key near expiry notification in Azure Key Vault to get alerts about upcoming expirations.

IaaS-specific considerations when using VMs

If you need to use IaaS VMs, some of the procedures and practices described earlier in this document might differ. The use of VMs provides more flexibility in configuration options, operating systems, driver access, low-level operating system access, and the kinds of software that you can install. The disadvantages are increased operational costs and the responsibility for tasks that are usually performed by the cloud provider when you use PaaS services.

Design considerations

- Individual VMs don't provide high availability, zone redundancy, or geo-redundancy.
- Individual VMs aren't automatically updated after you deploy them. For example, a deployed SQL Server 2019 on Windows Server 2019, won't automatically get

updated to a newer release.
- Services running in a VM need special treatment and additional tooling if you want to deploy and configure them via infrastructure as code.
- Azure periodically updates its platform. These updates might require VM reboots. Updates that require a reboot are usually announced in advance. See Maintenance for virtual machines in Azure and Handling planned maintenance notifications.

Design recommendations

- Avoid manual operations on VMs and implement proper processes to deploy and roll out changes.
  - Automate the provisioning of Azure resources by using infrastructure-as-code solutions like Azure Resource Manager (ARM) templates, Bicep, Terraform, or other solutions.

- Ensure that operational processes for deployment of VMs, updates, and backup and recovery are in place and properly tested. To test for resiliency, inject faults into the application, note failures, and mitigate those failures.

- Ensure that strategies are in place to roll back to the last known healthy state if a newer version doesn't function correctly.

- Create frequent backups for stateful workloads, ensure that backup tasks work effectively, and implement alerts for failed backup processes.

- Monitor VMs and detect for failures. The raw data for monitoring can come from a variety of sources. Analyze the causes of problems.

- Ensure that scheduled backups run as expected and that periodic backups are created as needed. You can use Backup center to get insights.

- Prioritize the use of Virtual Machine Scale Sets rather than VMs to enable capabilities like scale, autoscale, and zone redundancy.

- Prioritize the use of standard images from Azure Marketplace rather than custom images that need to be maintained.

- Use Azure VM Image Builder or other tools to automate build and maintenance processes for customized images as needed.

Beyond these specific recommendations, apply best practices for operational procedures for mission-critical application scenarios as appropriate.

Next step

Review the architecture pattern for mission-critical application scenarios:

Architecture pattern

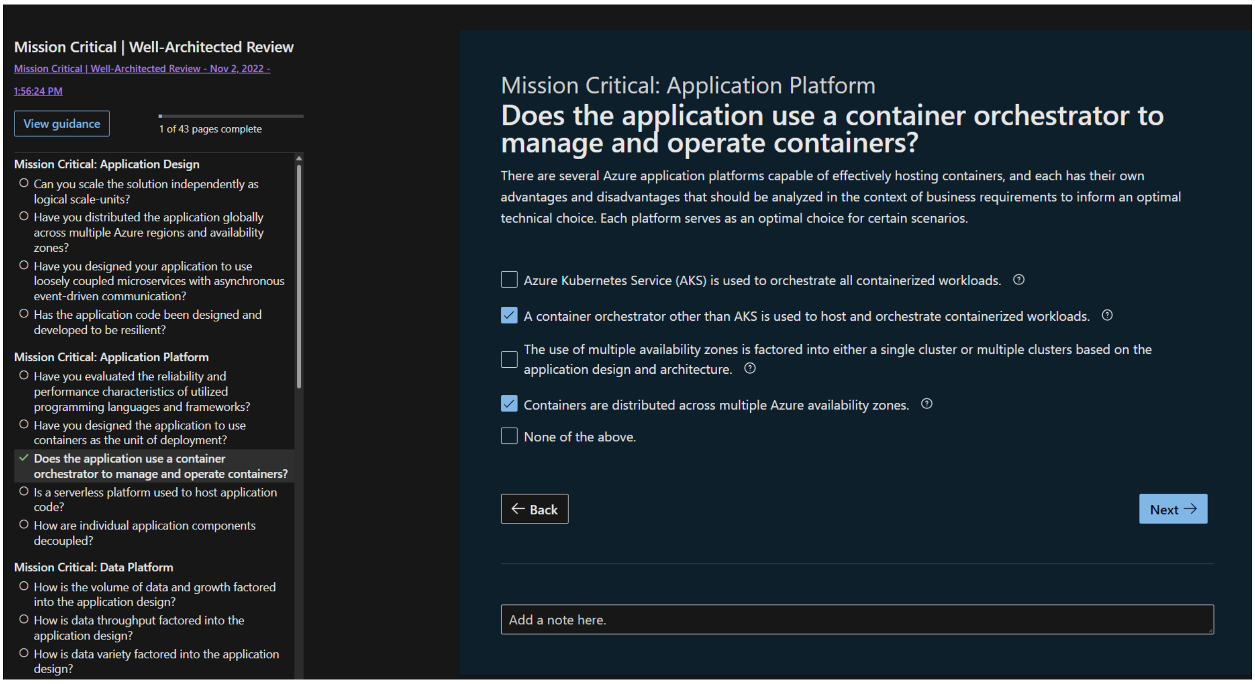
Well-Architected assessment for mission-critical workloads

Article • 03/15/2023

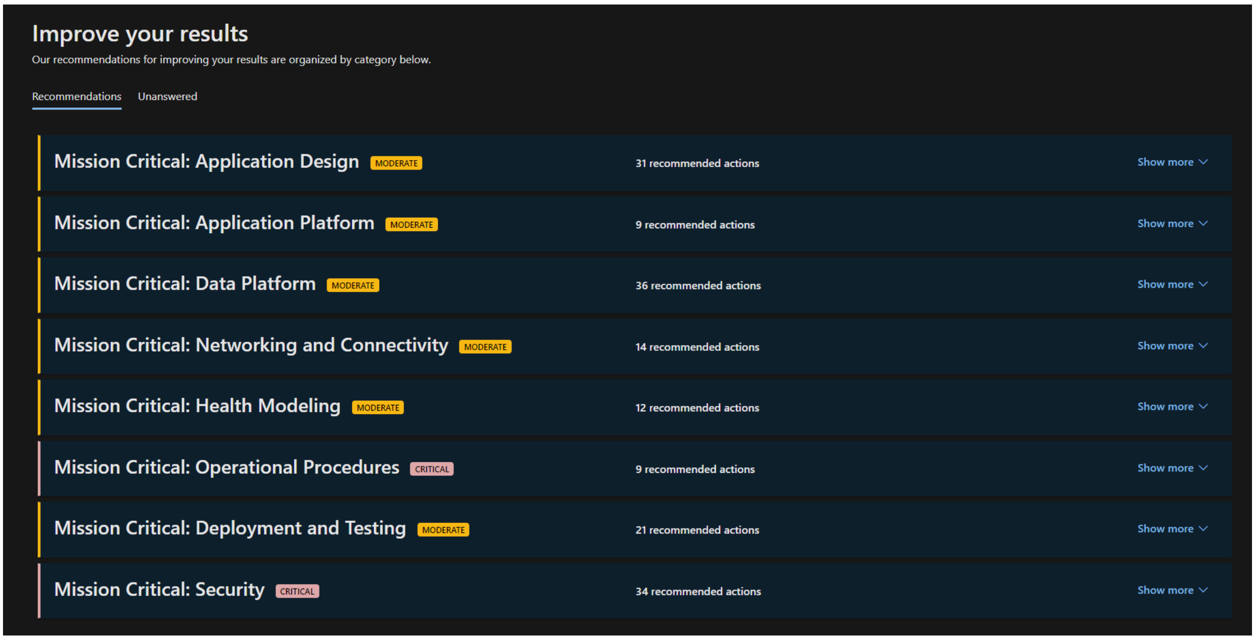
The Assessment is a review tool for self-assessing the readiness of your mission-critical workload in production. Working towards building a resilient mission critical architecture can be a complex process. The assessment is organized in a way that you'll be able to methodically check the alignment to the best practices for resiliency, reliability, and availability.

We recommend that the team doing the assessment is well versed in the architecture of the specific workload and has a strong understanding of cloud principles and patterns. These roles include, but aren't limited to, cloud architect, operators, DevOps engineer.

The assessment is a set of questions based on the mission-critical design methodology as a way of checking the foundational design choices of a workload's architecture and the end-to-end operational approach.



These questions are designed to help benchmark a workload's maturity and alignment to the recommended approach for operating a mission-critical workload. The outcome of the assessment is the delivery of technical recommendations and documentation that provide guidance on how to implement a highly reliable solution on Azure.



Mission-critical review tool

Next steps

See these reference architecture that describe design choices for production-ready implementations.

- Baseline architecture of an internet-facing application
- Baseline architecture of an internet-facing application with network controls

Carrier-grade workload documentation

In this series, learn about building highly reliable applications for carrier-grade workloads on Microsoft Azure. Mission-critical systems primarily focus on maximizing uptime and they exist in many industries. Within the telecommunications industry, they're referred to as carrier-grade systems.

Get started

OVERVIEW

What is carrier-grade?

Design principles

Design areas

CONCEPT

Fault tolerance

Data model

Health modeling

Testing and validation

Carrier-grade workloads on Azure

Article • 03/20/2023

Mission-critical systems primarily focus on maximizing uptime and they exist in many industries. Within the telecommunications industry, they're referred to as *carrier-grade systems*. These systems are developed due to one or more of the following drivers:

- Minimizing loss of life or injury.
- Meeting regulatory requirements on reliability to avoid paying fines.
- Optimizing service to customers to minimize churn to competitors.
- Meeting contractual Service Level Agreements (SLAs) with business or government customers.

This series of articles applies the design methodology for mission-critical workloads to inform prescriptive guidance for building and operating a highly reliable, resilient, and available telecommunication workload on Azure.

> **Note**
>
> The articles within this series focus on providing additional mission-critical considerations when designing for 99.999% ('5 9s') levels of reliability within the telecommunications industry.

What is a carrier-grade workload?

The term *workload* refers to a collection of application resources that support a common business goal or the execution of a common business process, with multiple services, such as APIs and data stores, working together to deliver specific end-to-end functionality.

The term *mission-critical* refers to a criticality classification where a significant financial cost (business-critical) or human cost (safety-critical) is associated with unavailability or underperformance.

A *carrier-grade workload* pivots on both business-critical and safety-critical, where there's a fundamental requirement to be operational with only minutes or even seconds of downtime per calendar year. Failure to achieve this uptime requirement can result in extensive loss of life, incur significant fines, or contractual penalties.

The *operational* aspect of the workload includes how reliability is measured and the targets that it must meet or exceed. Highly reliable systems typically target 99.999% uptime (commonly referred to as '5 9s') or 0.001% downtime in a year (approximately 5 minutes). Some systems target 99.9999% uptime, or 30 seconds downtime per year, or even higher levels of reliability. This covers all forms and causes of outage – scheduled maintenance, infrastructure failure, human error, software issues and even natural disaster.

Although the platform used has evolved from dedicated, proprietary hardware through commercial, off-the-shelf hardware to OpenStack or VMware clouds, Telecommunication companies consistently deliver services achieving ≤ 5 minutes of downtime per year, and in many cases, achieve ≤ 30 seconds of downtime due to unscheduled outages.

What are the common challenges?

Building carrier-grade workloads is a challenge for these main reasons:

Lift and shift approach

Telecommunication companies have well-architected applications that deliver the expected behavior on their existing infrastructure. However, care should be taken before assuming that porting these applications *as is* to a public cloud infrastructure won't impact their resiliency.

The existing applications make a set of assumptions about their underlying infrastructure, which are unlikely to remain true when moving from on-premises to public cloud. The architect must check that they still hold and adjust infrastructure and application design to accommodate the new reality. The architect should also look for opportunities where the new infrastructure removes limitations that existed on-premises.

For example, upgrade of on-premises systems must happen in place because it's not viable to maintain sufficient hardware to create a new deployment alongside and slowly transition in a safe manner. This upgrade path generates a host of requirements for how upgrades and rollbacks are managed. These requirements lead to complexity and mean that upgrades are infrequent and only permitted in carefully managed maintenance windows.

However, in public cloud, it's reasonable to create a new deployment in parallel with the existing deployment. This process creates the opportunity for major simplifications in the application's operational design and improvements in the user's experience, and expectations.

Monolithic solutions

Experience across a range of mission-critical industries shows that it isn't realistic to attempt to construct a monolithic solution which will achieve the desired levels of availability. There are just too many potential sources of failure in a complex system. Instead, successful solutions are composed of individual independent and redundant elements. Each unit has relatively basic availability (typically ~99.9%), but when combined together the total solution achieves the necessary availability goals. The art of carrier-grade engineering then becomes ensuring that the only dependencies common to the redundant elements are those which are absolutely necessary since they exert the most significant influence on overall availability, often orders of magnitude greater than anything else in the design.

Only building zonal redundancy

Using Microsoft Azure Availability Zones is the basic choice for reducing the risk of outage due to hardware failure or localized environmental issues. However, it isn't enough to achieve carrier-grade availability, mainly for these reasons:

- Availability Zones (AZs) are designed so that the network latency between any two zones in a single region is ≤ 2 ms. AZs can't be widely and geographically dispersed. So, the AZs share a correlated risk of failure due to natural disasters, such as flooding or massive power outages, which could disable multiple AZs within a region.

- Many Azure services are explicitly designed to be zone-redundant so that the applications using them don't need explicit logic to benefit from the availability gain. This redundancy function within the service requires collaboration between the elements in each zone. There's often an unavoidable risk of software failure in one zone that causes correlated failures in other zones. For example, any issue with a secret or certificate used with the zone redundant service could impact all AZs simultaneously.

What are the key design areas?

When designing a carrier-grade workload, consider the following areas:

Design area	Description

Design area	Description
Fault tolerance	Application design must allow for unavoidable failures so that the application as a whole can continue to operate at some level. The design must minimize points of failure and take a federated approach.
Data model	The design must address the consistency, availability, and partition tolerance needs of the database. Carrier Grade availability requires that the application is deployed across multiple regions. This deployment requires careful thought about how the application's data will be managed across those regions.
Health modeling	An effective health model provides observability of all components, platform and application, so that issues can be quickly detected and response is ready either through self-healing or other remediations.
Testing and validation	The design and implementation of the application must be thoroughly tested. In addition, the integration and deployment of the application as a whole solution must be tested.

Next step

Start by reviewing the design principles for carrier-grade application scenarios.

Design principles

Design principles of a carrier-grade workload on Azure

Article • 01/11/2023

Carrier grade workload must be designed as per the guiding principles of the Well-Architected Framework quality pillars:

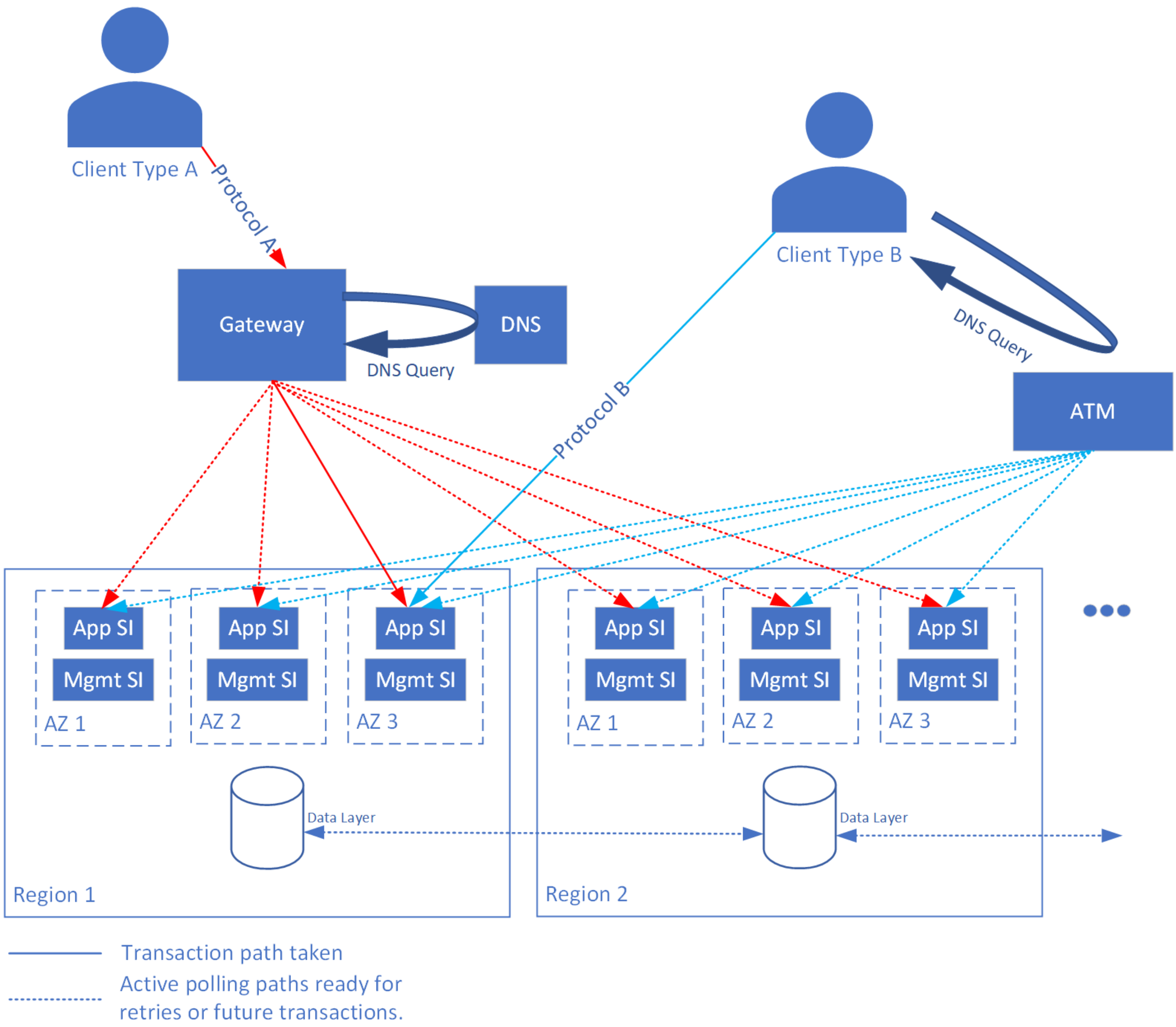
- Reliability
- Performance Efficiency
- Operational Excellence
- Security
- Cost Optimization

This article describes the carrier-grade design principles that resonate and extend the mission-critical design principles. These collective principles serve as a road map for subsequent design decisions across the critical design areas. We highly recommend that you get to know these principles to better understand their effects and the trade-offs associated with non-adherence.

There are obvious cost tradeoffs associated with introducing greater reliability, which should be carefully considered in the context of workload requirements.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected carrier-grade workload** series. If you aren't familiar with this series, we recommend you start with **What is a carrier-grade workload**?

Keep this high-level architecture model in mind when considering these points.



Assume failure

Start from the assumption that everything can, and will fail. Application design must allow for these failures with fault tolerance so that an application can continue to operate at some level.

- Minimize single points of failure and implement a federated approach.
- Deploy the application across multiple regions with proper data management across those regions, allowing for the impacts of CAP theorem.
- Detect issues automatically and respond within seconds. For more information, see health monitoring.
- Test the full solution including the application implementation, platform integration, and deployment. This testing should include chaos testing on production systems to avoid testing bias.

Share nothing

Share nothing is a common and straightforward approach to achieve high availability. Use this approach when an application can be serviced by multiple, distinct elements, which are interchangeable. The individual elements must have a well-understood availability metric, but it doesn't need to be high. However, the elements must be combined in a way to remain independent, with no shared infrastructure or dependencies.

To share nothing is often impossible. To start from the position that nothing *should* be shared, and only add in the smallest possible set of shared dependencies, should result in an optimal solution.

Example

Given a single system that has six hours of downtime per year (around 3.5*9s), a solution that combines four systems where the periods of downtime are uncorrelated will experience less than 30s of downtime per year. As soon as those four systems rely on a common service, such as global DNS, their downtime is no longer uncorrelated. The resulting downtime will be higher.

Next step

Review the fault tolerance design area for carrier-grade workloads.

Design area: Fault tolerance

Fault tolerance for carrier-grade workloads

Article • 01/06/2023

Telecommunication companies have shown that it's possible to implement applications that meet and exceed carrier-grade availability requirements, whilst running on top of unreliable elements. Companies exceed these requirements through *fault tolerance*, which includes the following aspects:

- Combination of independent, non-highly available elements.
- Traffic management failure response mechanisms.
- Repair and capacity-recovery and failure response mechanisms.
- Use of highly available cross-element databases.

When designing for carrier grade reliability and resiliency, assume that every component can and *will* fail. The design will require a layered approach to failure resolution. Part of validating the design is creating a quantitative probabilistic model of the various failure modes which clearly identifies the key dependencies that the application has, and shows that the application can achieve the necessary availability given those dependencies meet their own Service Level Objectives (SLOs). This model should be retained and continuously validated after development and in production to assure that the test and live data match what the model predicts.

High availability through combination

To reach high availability, take independent elements, which aren't highly available, and combine them so that they remain independent entities. The probability of multiple elements failing together is much lower than the probability of failure of any single element. We define this process as the Share Nothing architectural style.

Traffic management failure response

When individual elements fail, or there are connectivity issues, the system's traffic management layer has various ways it can respond to maintain overall service. The solution should implement as many of the following mechanisms as possible or necessary to achieve availability. The first two mechanisms rely on specific behavior in the client, which may not always be an option:

1. **Instantaneous client retry to alternate target**—On failure, or after no response from a retry, the client can instantly retry the request with an alternative element that's considered likely to succeed, as opposed to submitting a new DNS query to identify an alternative.

2. **Client-based health lists**—Maintains lists of elements, which are *known-healthy* and *known-unhealthy*. Requests are always sent to a known-healthy element, unless that list is empty. If empty, the known-unhealthy elements are tried.

3. **Server-based polling**—System-based distribution mechanisms, such as DNS or Azure Traffic Manager (ATM), implement their own liveness detection and monitoring to enable routing around unhealthy elements.

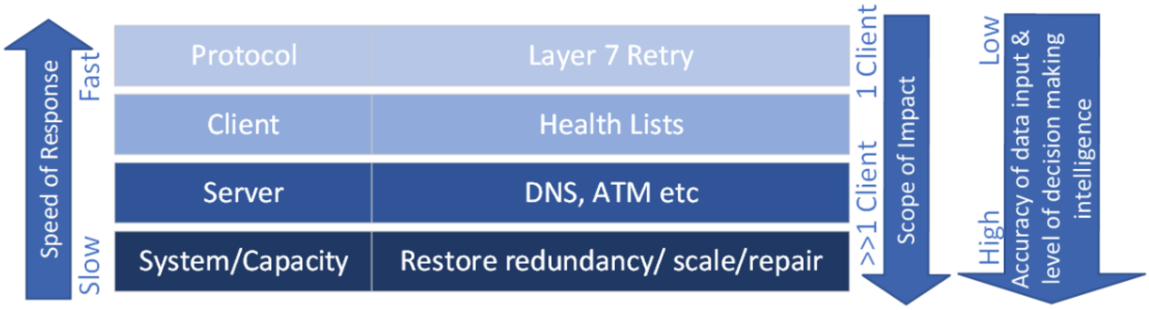
Repair and recovery failure response

Traffic management responses can work around failures, but where the failure is long-lived or permanent, the defective element must be repaired or replaced. There are two main choices here:

1. **Restoring redundancy**—Failure of an element reduces overall system capacity. System design should allow for this capacity reduction through provisioning redundant capacity; however, multiple overlapping failures will lead to true outages. There must be an automated process that detects the failure and adds capacity to restore redundancy to its normal levels. The impact can be determined from the failure rate analysis. In many cases, automatically restoring the redundancy level can increase the acceptable downtime of any individual element.

2. **(Optional) Repairing the failed element**—The solution may need to include a mechanism that detects the failure and attempts to repair the failed element in place. This solution may not apply for cases where the process of restoring redundancy instantiates a new element to replace the failed one, and terminates and decommissions the failed element.

The following diagram shows how the two modes of failure response cooperate to provide service-level fault tolerance:



Failure rate analysis

No amount of effort leads to a perfect system, so start with the assumption that everything can and will fail. Consider how the solution, as a whole, will behave. Failure rate analysis starts with the lower-level individual systems and combines the results together to model the full solution.

The analysis is typically done using Markov modeling, which considers all possible failure modes. For each failure mode, consider the following factors:

- Rate
- Duration and resolution time
- Probability of correlated failure (what is the chance that a failure in service X causes a failure in service Y)

The outcome is an estimate of the system availability and informs consideration of the *blast radius*. The blast radius is the set of things that won't work given a particular failure. Good design aims to limit the scope and severity of that set, since failure is going to happen. For example, a failure that blocks creation of new users, but doesn't impact existing ones is less concerning than a failure that stops service for all users.

Failure rate analysis provides an estimate of the overall system-level availability. The analysis identifies the critical dependencies on which that availability relies. Where failure rate analysis indicates that system availability falls short, it also provides specific, quantitative insights on where improvements are needed and to what extent.

For more details on failure mode analysis in Azure, reference [Failure mode analysis for Azure applications](#).

Cascading failures and overload

Carrier-grade application design must pay careful attention to the risks of cascading failures, where failure of one element leads to failure of other elements, often due to overload. Cascading failures aren't unique to carrier-grade applications, but the reliability and the response time demand graceful degradation and automated recovery.

Next step

Review the considered data model design area for carrier-grade workloads.

Design area: Data model

Data modeling for carrier-grade workloads

Article • 09/26/2022

Enterprise applications deployed in a single region can typically ignore this model of application design and safely delegate responsibility to the database layer to make data reliably available. This behavior isn't the case for carrier-grade applications, which must be deployed across multiple regions. Multi-region deployment forces the application architect to consider the compromises they're willing to accept for their data.

CAP theorem

Databases can provide three key properties for the data they manage for an application, known as CAP:

- Consistency: A data read returns the most recent write.
- Availability: Every request receives a valid response.
- Partition tolerance: The system continues to operate despite delay or total loss of communication between elements.

The CAP theorem states that a database layer can't provide all three of these properties for the same data at the same time in the presence of network partitions. The architect needs to make explicit design decisions about which of the CAP properties to favor under which conditions, and how to deal with the edge cases.

According to the CAP theorem, any database can only guarantee two out of three possible properties for the same data at the same time in the presence of network partition.

Multi-region deployment means partition tolerance becomes significant. In most cases, carrier-grade architects prioritize partition tolerance and availability over consistency. For each type of data, the architect must consider what tradeoffs they're willing to make, considering the edge cases.

For example, consider the database of system users. Is it acceptable for the user database to drop to read-only if there's a network partition? This behavior prioritizes consistency and read availability over write availability. This prioritization may not be suitable if it's unacceptable for a user to access an isolated site after their permissions are revoked elsewhere. The described scenario would require all database access to be blocked if there's a partition, which prioritizes write availability over read availability.

> ⓘ **Note**
>
> The compromises made can be different for different databases within the same application, since the databases are likely to have different usage profiles.

Where consistency is the compromise, the application must cope with data inconsistency artifacts by using conflict-free replicated data types (CRDTs). The use of CRDTs requires discipline in the application design and implementation. Their use means data merges following partition events can be handled automatically by the data layer without human intervention or complex application-level logic.

> ⓘ **Important**
>
> More details on data platform choices for your mission-critical workload is available **here**

The architect must also understand the tradeoffs in the data model, which were made within the dependent services. Those tradeoffs impact the service delivered to their application because those tradeoffs may not align with the application-level requirements.

Next step

Review the health modeling design area for carrier-grade workloads.

Design area: Health modeling

Health modeling for carrier-grade workloads

Article • 01/11/2023

High availability requires careful health monitoring to automatically detect and respond to issues within seconds. This monitoring requires built-in telemetry of key dependencies to reliably detect the failure. The application itself requires additional telemetry (Service Level Indicators) which accurately report the health of the application in a way that is perceived by users of the application. Evaluation against SLOs may be necessary.

The failure rate analysis and general health modeling of the application should generate clear metrics indicative of the service and health of its constituent elements. These metrics must be included in the design so that the true service availability can be monitored. By including metrics, you can track the most useful leading indicators to trigger the automated failure responses and to generate the necessary alerts for human intervention.

> ⓘ **Important**
>
> More details on how to build a health model for your mission-critical workload are available **here**.

Management and monitoring

Monitoring and management require the following thought processes:

- How will the application handle bugs in the framework?
- How is the application upgraded?
- What actions should be taken during an incident?

For example, a solution may rely on Azure DevOps (ADO) to host its Git repository for all configuration. If the Azure region hosting that ADO repo fails, the recovery time is two hours. If the solution is deployed in the same region, it's not possible to modify configuration to add capacity elsewhere for that entire two hour period. As a result, the application architect must consider correlated failure modes for key services, such as:

- Azure Traffic Manager
- Azure Key Vault

- Azure Kubernetes Service

Correlated failure modes for these key services may be a necessary part of the application-level response to failure. It's vital to create control planes which aren't impacted by the same application failure.

The management tooling required to issue diagnosis and troubleshooting must be the same as tooling used for normal day-to-day operations tasks. Similar tooling ensures that it's familiar and proven to work. Similar tooling also maximizes the users' familiarity with the user interface and process steps. Requiring operators to switch to a different tool set to resolve a high-pressure outage isn't conducive to identifying and resolving the issue effectively.

Federated model

A highly available application or service must have a highly available management and monitoring infrastructure built using the same well-architected principles of federation and fault tolerance. Infrastructures built on these well-architected principles ensure individual regions can be self sufficient if disconnected.

If there's a disconnect event, the system degenerates into individually functioning islands, instead of using a primary/backup system. The federated model is flexible and resilient, and automatically adapts to partition and reconnection events.

For example, logs and metrics are stored in the Availability Zone (AZ) where they're generated. A query of metrics uses an opaque process of federated search to query the metric stores in all reachable AZs. It comes down to the requirements of the specific application about what level of logs, metrics, and alarms data should be replicated to other regions. Typically, alarms should be replicated, but there may be insufficient justification to replicate logs and metrics.

Health and unhealth metrics

Internal metrics are useful as *unhealth* metrics. These metrics reliably indicate the presence of an issue, but the reverse isn't true. No evidence of poor health isn't evidence of good health, as the customer perceives health.

For example, a DNS issue indicates requests aren't arriving at the database service. The DNS error doesn't affect the database read-success metric because this metric isn't seeing any errors. However, the end user perceives a total outage because they aren't able to access the database. At least a portion of *health* metrics must be measured externally, so that these metrics include everything the end user will experience.

Monitoring and tracing

The support team's ability to detect, diagnose, and resolve issues is an important part of delivering a highly available application. To ensure success, the monitoring and tracing element must deliver high levels of visibility, so that the one in a thousand type events can be found and resolved.

A tracing solution that only logs 0.1% of requests only has a one in one million chance of recording such events, which means that diagnosis and resolution are highly unlikely. Yet, failure to resolve such issues will have a meaningful impact on availability.

Next step

Review the testing and validation design area for carrier-grade workloads.

Design area: Testing and validation

Testing and validation for carrier-grade workloads

Article • 09/26/2022

Continuous testing and validation can detect and help resolve issues before they become potentially life threatening. Consider well-known testing methodologies such as chaos testing. Testing should be conducted for the lifetime of the application because the deployment environment is complex and multi-layered.

> ⓘ **Important**
>
> More details on how to implement continuous validation for your mission-critical workload is available **here**.

Also, supportability must be strong throughout the application lifetime. Highly available systems rely on high quality support teams able to rapidly respond to and resolve issues in the field, conduct root cause analysis and look for systematic design flaws.

Proving that an application is well architected requires testing, ideally use a chaos testing framework to avoid testing bias. This methodology simulates failures of all dependent elements. Robust and regular testing should both prove the design and validate the original failure mode analysis.

A warning flag should be raised for any application or service for which the redundancy or resiliency measures can't be tested because it's considered *too risky*.

If redundancy and resiliency measures aren't tested, then the only valid assumption, from a safety-critical point of view, is that these measures aren't going to work when needed. Using common paths for software upgrades, configuration updates, and fault recovery, for example, provide a good mechanism for validating that measures will work.

Human error

Experience from Telcos is that as much as 60% of all outages are actually the result of human error. A well-architected application recognizes this and seeks to compensate. Here are some suggested approaches, but the list is not exhaustive, and what is applicable to a given workload needs to be considered on a case-by-case basis.

- Maximizing use of automation avoids human operators having to enter long and complex commands or conduct repetitive operations across multiple elements.

However, care must be taken to consider the blast radius, as there is a risk of automation actually magnifying the effect of a configuration error, allowing it to roll out across a global network in seconds. Strong checks and balances such as decision gates requiring human approval before proceeding to the next step are advised.

- Leveraging syntax checkers and simulation tools minimize the chance of errors or unforeseen side effects from changes making their way into widespread production.
- Use of carefully controlled canary deployments ensure that the effect of changes in full production can be observed and validated at limited scope.
- Ensuring that the management interfaces and processes needed for fault recovery are the same as those used in day-to-day operation avoid operators being confronted with unfamiliar screens and barely used method of procedures (MOPs) at times of peak stress.

Clients

Common client libraries are also part of the end-to-end system and need equivalent analysis and testing. Software issues in common client code that simultaneously impacts a proportion of the system clients will impact overall availability in the same way as application server-side issues.

Next step

Revisit the five pillars of architectural excellence to form a solid foundation for your carrier-grade workloads.

Azure Well-Architected Framework

Overview of a hybrid workload

Article • 03/20/2023

Customer workloads are becoming increasingly complex, with many applications often running on different hardware across on-premises, multicloud, and the edge. Managing these disparate workload architectures, ensuring uncompromised security, and enabling developer agility are critical to success.

Azure uniquely helps you meet these challenges, giving you the flexibility to innovate anywhere in your hybrid environment while operating seamlessly and securely. The Well-Architected Framework includes a hybrid description for each of the five pillars: cost optimization, operational excellence, performance efficiency, reliability, and security. These descriptions create clarity on the considerations needed for your workloads to operate effectively across hybrid environments.

Adopting a hybrid model offers multiple solutions that enable you to confidently deliver hybrid workloads: run Azure data services anywhere, modernize applications anywhere, and manage your workloads anywhere.

Extend Azure management to any infrastructure

> **Tip**
>
> Applying the principles in this article series to each of your workloads will better prepare you for hybrid adoption. For larger or centrally managed organizations, hybrid and multicloud are commonly part of a broader strategic objective. If you need to scale these principle across a portfolio of workloads using hybrid and multicloud environments, you may want to start with the Cloud Adoption Framework's **hybrid and multicloud scenario and best practices**. Then return to this series to refine each of your workload architectures.

Use *Azure Arc enabled infrastructure* to extend Azure management to any infrastructure in a hybrid environment. Key features of Azure Arc enabled infrastructure are:

- **Unified Operations**
  - Organize resources such as virtual machines, Kubernetes clusters and Azure services deployed across your entire IT environment.
  - Manage and govern resources with a single pane of glass from Azure.

- Integrate with Azure Lighthouse for managed service provider support.

- **Adopt cloud practices**
  - Easily adopt DevOps techniques such as infrastructure as code.
  - Empower developers with self-service and choice of tools.
  - Standardize change control with configuration management systems, such as GitOps and DSC.

Run Azure services anywhere

Azure Arc allows you to run Azure Services anywhere. This allows you to build consistent hybrid and multicloud application architectures by using Azure services that can run in Azure, on-premises, at the edge, or at other cloud providers.

Run Azure data services anywhere

Use *Azure Arc enabled data services* to run Azure data services anywhere to support your hybrid workloads. Key features of Azure Arc enabled data services are:

- Run Azure data services on any Kubernetes cluster deployed on any hardware.
- Gain cloud automation benefits, always up-to-date innovation in Azure data services, unified management of your on-premises and cloud data assets with a cloud billing model across both environments.
- Azure SQL Database and Azure PostgreSQL Hyperscale are the first set of Azure data services that are Azure Arc enabled.

Run Azure Application services anywhere

Use *Azure Arc enabled Application services* to run Azure App Service, Functions, Logic Apps, Event Grid, and API Management anywhere to support your hybrid workloads. Key features of Azure Arc enabled application services are as follows:

- Web Apps - Azure App Service makes building and managing web applications and APIs easy, with a fully managed platform and features like autoscaling, deployment slots, and integrated web authentication.
- Functions - Azure Functions makes event-driven programming simple, with state-of-the-art autoscaling, and with triggers and bindings to integrate with other Azure services.
- Logic Apps - Azure Logic Apps produces automated workflows for integrating apps, data, services, and backend systems, with a library of more than 400 connectors.

- Event Grid - Azure Event Grid simplifies event-based applications, with a single service for managing the routing of events from any source to any destination.
- Azure API Management gateway - Azure API Management provides a unified management experience and full observability across all internal and external APIs.

Modernize applications anywhere

Use the *Azure Stack family* to modernize applications without ever leaving the datacenter. Key features of the Azure Stack family are:

- Extend Azure to your on-premises workloads with Azure Stack Hub. Build and run cloud apps on premises, in connected or disconnected scenarios, to meet regulatory or technical requirements.
- Use Azure Stack HCI to run virtualized workloads on premises and easily connect to Azure to access cloud management and security services.
- Build and run your intelligent edge solutions on Azure Stack Edge, an Azure managed appliance to run machine learning models and compute at the edge to get results quickly—and close to where data is being generated. Easily transfer the full data set to Azure for further analysis or archive.

Manage workloads anywhere

Use *Azure Arc management* to extend Azure management to all assets in your workloads, regardless of where they are hosted. Key features of Azure Arc management are:

- **Adopt cloud practices**
  - Easily adopt DevOps techniques such as infrastructure as code.
  - Empower developers with self-service and choice of tools.
  - Standardize change control with configuration management systems, such as GitOps and DSC.

- **Scale across workloads with Unified Operations**
  - Organize resources such as virtual machines, Kubernetes clusters and Azure services deployed across your entire IT environment.
  - Manage and govern resources with a single pane of glass from Azure.
  - Integrate with Azure Lighthouse for managed service provider support.

Next steps

Cost optimization

Cost optimization in a hybrid workload

Article • 11/30/2022

A key benefit of hybrid cloud environments is the ability to scale dynamically and back up resources in the cloud, avoiding the capital expenditures of a secondary datacenter. However, when workloads sit in both on-premises and cloud environments, it can be challenging to have visibility into the cost. With Azure's hybrid technologies, you can define policies and constraints for both on-premises and cloud workloads with Azure Arc. By utilizing Azure Policy, you're able to enforce organizational standards for your workload and the entire IT estate.

Azure Arc helps minimize or even eliminate the need for on-premises management and monitoring systems, which reduces operational complexity and cost, especially in large, diverse, and distributed environments. This helps offset additional costs associated with Azure Arc-related services. For example, advanced data security for Azure Arc enabled SQL Server instance requires Microsoft Defender for Cloud functionality of Microsoft Defender for Cloud, which has pricing implications.

Other considerations are described in the Principles of cost optimization section in the Microsoft Azure Well-Architected Framework.

Workload definitions

Define the following for your workloads:

- **Monitor cloud spend with hybrid workloads**. Track cost trends and forecast future spend with dashboards in Azure for your on-prem data estates with Azure Arc.
- **Keep within cost constraints**.
  - Create, apply, and enforce standardized and custom tags and policies.
  - Enforce run-time conformance and audit resources with Azure Policy.
- **Choose a flexible billing model**. With Azure Arc enabled data services, you can use existing hardware with the addition of an operating expense (OPEX) model.

Functionality

For budget concerns, you get a considerable amount of functionality at no cost that you can use across all of your servers and cluster with Azure Arc enabled servers. You can turn on additional Azure services to each workload as you need them, or not at all.

- **Free Core Azure Arc capabilities**

- Update, management
- Search index
- Group, tags
- Portal
- Templates, extensions
- RBAC, subscriptions

- **Paid-for Azure Arc enabled attached services**
  - Azure policy
  - Azure monitor
  - Defender for Cloud – Standard
  - Microsoft Sentinel
  - Backup
  - Config and change management

Tips

- **Start slow**. Light up new capabilities as needed. Most of Azure Arc's resources are free to start.
- **Save time with unified management** for your on-premises and cloud workloads by projecting them all into Azure.
- **Automate and delegate** remediation of incidents and problems to service teams without IT intervention.

Azure Architecture Center (AAC) resources related to hybrid cost

- Manage configurations for Azure Arc enabled servers
- Azure Arc hybrid management and deployment for Kubernetes clusters
- Optimize administration of SQL Server instances in on-premises and multi-cloud environments by leveraging Azure Arc
- Disaster Recovery for Azure Stack Hub virtual machines
- Build high availability into your BCDR strategy
- Use Azure Stack HCI switchless interconnect and lightweight quorum for Remote Office/Branch Office
- Archive on-premises data to cloud

Infrastructure Decisions

Azure Stack HCI can help in cost-savings by using your existing Hyper-V and Windows Server skills to consolidate aging servers and storage. Azure Stack HCI pricing follows the monthly subscription billing model, with a flat rate per physical processor core in an Azure Stack HCI cluster.

Use Azure Stack HCI to modernize on-prem workloads with hyperconverged infra. Azure Stack HCI billing is based on a monthly subscription fee per physical processor core, not a perpetual license. When customers connect to Azure, the number of cores used is automatically uploaded and assessed for billing purposes. Cost doesn't vary with consumption beyond the physical processor cores. This means that more VMs don't cost more, and customers who are able to run denser virtual environments are rewarded.

If you are currently using VMware, you can take advantage of cost savings only available with Azure VMware Solution. Easily move VMware workloads to Azure and increase your productivity with elasticity, scale, and fast provisioning cycles. This will help enhance your workloads with the full range of Azure compute, monitor, backup, database, IoT, and AI services.

Lastly, you can slowly begin migrating out of your datacenter and use Azure Arc while you're migrating to project everything into Azure.

Capacity planning

Check out our checklist under the Cost Optimization pillar in the Well-Framework to learn more about capacity planning, and build a checklist to design cost-effective workloads.

- Define SLAs
- Determine regulatory needs

Provision

One advantage of cloud computing is the ability to use the PaaS model. And in some cases, PaaS services can be cheaper than managing VMs on your own. Some workloads cannot be moved to the cloud though for regulatory or latency reasons. Therefore, using a service like Azure Arc enabled services allows you to flexibly use cloud innovation where you need it by deploying Azure services anywhere.

Click the following links for guidance in provisioning:

- Azure Arc pricing
- Azure Arc Jumpstart for templates (in GitHub)

- Azure Stack HCI pricing
  - Azure Stack HCI can reduce costs by saving in server, storage, and network infrastructure.
- Azure VMware Solution pricing - Run your VMware workloads natively on Azure
  - Run your VMware workloads natively on Azure.
- Azure Stack Hub pricing

Monitor and optimize

Treat cost monitoring and optimization as a process, rather than a point-in-time activity. You can conduct regular cost reviews and forecast the capacity needs so that you can provision resources dynamically and scale with demand.

- Managing the Azure Arc enabled servers agent
  - Bring all your resources into a single system so you can organize and inventory through a variety of Azure scopes, such as Management groups, Subscriptions, and Resource Groups.
  - Create, apply, and enforce standardized and custom tags to keep track of resources.
  - Build powerful queries and search your global portfolio with Azure Resource Graph.
- With Azure Stack HCI
  - Costs for datacenter real estate, electricity, personnel, and servers can be reduced or eliminated.
  - Costs are now part of OPEX, which can be scaled as needed.

Next steps

Operational excellence

Operational excellence in a hybrid workload

Article • 11/30/2022

Operational excellence consists of the operations processes that keep a system running in production. Applications must be designed with DevOps principles in mind, and deployments must be reliable and predictable. Use monitoring tools to verify that your application is running correctly and to gather custom business telemetry that will tell you whether your application is being used as intended.

Use *Azure Arc enabled infrastructure* to add support for cloud Operational Excellence practices and tools to any environment. Be sure to utilize reference architectures and other resources from this section that illustrate applying these principles in hybrid and multicloud scenarios. The architectures referenced here can also be found in the Azure Architecture Center, Hybrid and Multicloud category.

Build cloud native apps anywhere, at scale

To keep your systems running, many workload teams have architected and designed applications where components are distributed across public cloud services, private clouds, data centers, and edge locations. With Azure Arc enabled Kubernetes, you can accelerate development by using best in class application services with standardized deployment, configuration, security, and observability. One of the primary benefits of Azure Arc is facilitating implementation of DevOps principles that apply established development practices to operations. This results in improved agility, without jeopardizing the stability of IT environment.

- Centrally code and deploy applications confidently to any Kubernetes distribution in any location.
- Centrally manage and delegate access for DevOps roles and responsibilities.
- Reduce errors with consistent configuration and policy-driven deployment and operations for applications and Kubernetes clusters.
- Delegate access for DevOps roles and responsibilities through Azure RBAC.
- Reduce errors with consistent policy driven deployment and operations through GitHub and Azure Policy.

Connect Kubernetes clusters to Azure and start deploying using a GitOps model

GitOps relies on a Git repository to host files that contain the configuration representing the expected state of a resource. An agent running on the cluster monitors the state of the repository and, when there is a change on the repository, the agent pulls the changed files to the cluster and applies the new configuration.

In the context of Azure Arc enabled Kubernetes clusters, a Git repository hosts a configuration of a Kubernetes cluster, including its resources such as pods and deployments. A pod or a set of pods running on the cluster polls the status of the repository and, once it detects a change, it pulls and applies the new configuration to the cluster.

Azure Arc enabled Kubernetes clusters rely on Flux, an open-source GitOps deployment tool to implement the pods responsible for tracking changes to the Git repository you designate and applying them to the local cluster. In addition, the containerized Flux operator also periodically reviews the existing cluster configuration to ensure that it matches the one residing in the Git repository. If there is a configuration drift, the Flux agent remediates it by reapplying the desired configuration.

Each association between an Azure Arc enabled Kubernetes cluster configuration and the corresponding GitOps repository resides in Azure, as part of the Azure Resource Manager resource representing the Azure Arc enabled Kubernetes clusters. You can configure that association via traditional Azure management interfaces, such as the Azure portal or Azure CLI. Alternatively, you can use Azure Policy to automate this process, allowing you to apply it consistently to all resources in an entire subscription or individual resource groups you designate.

Modernize applications anywhere with Azure Kubernetes Service on Azure Stack HCI

If you are looking for a fully managed Kubernetes solution on-premises in your datacenters and/or edge locations, AKS on Azure Stack HCI is a great option. Azure Kubernetes Service on Azure Stack HCI is an on-premises implementation of Azure Kubernetes Service (AKS), which automates running containerized applications at scale. Azure Kubernetes Service is now in preview on Azure Stack HCI and Windows Server 2019 Datacenter, making it quicker to get started hosting Linux and Windows containers in your datacenter.

AKS clusters on Azure Stack HCI can be connected to Azure Arc for centralized management. Once connected, you can deploy your applications and Azure data services to these clusters and extend Azure services such as Azure Monitor, Azure Policy and Microsoft Defender for Cloud.

Azure Stack HCI use cases

- **Modernize your high-performance workloads and containerized applications**
  - Use Azure Stack HCI to enable automated deployment, scaling and management of containerized applications by running a Kubernetes cluster on your hyperconverged infrastructure.
  - Deploy AKS on Azure Stack HCI using Windows Admin Center or PowerShell.

- **Deploy and manage workloads in remote and branch sites**
  - Use Azure Stack HCI to deploy your container-built edge workloads, and essential business applications in highly available virtual machines (VMs).
  - Bring efficient application development and deployment to remote locations at the right price by leveraging switchless deployment and 2 node clusters.
  - Get a global view of your system's health using Azure Monitor.

- **Upgrade your infrastructure for remote work using VDI**
  - Bring desktops on-premises for low latency and data sovereignty enabling remote work using a brokerage service like Microsoft Remote Desktop Services. With Azure Stack HCI you can scale your resources in a simple predictable way. Provide a secure way to deliver desktop services to a wide range of devices without allowing users to store data locally or upload data from those local devices.

Resources and architectures related to Operational Excellence

The introduction of cloud computing had a significant impact on how software is developed, delivered, and run. With *Azure Arc enabled infrastructure* and Azure Arc components like Azure Arc enabled Kubernetes and Azure Arc enabled data services it becomes possible to design cloud native applications with a consistent set of principles and tooling across public cloud, private cloud, and the edge.

Click the following links for architecture details and diagrams that enable application design and DevOps practices consistent with Operational excellence principles.

Application design

- Azure Arc hybrid management and deployment for Kubernetes clusters
- Run containers in a hybrid environment
- Managing K8 clusters outside of Azure with Azure Arc

- Optimize administration of SQL Server instances in on-premises and multi-cloud environments by leveraging Azure Arc
- Azure Data Studio dashboards
- microsoft/azure_arc: Azure Arc environments bootstrapping for everyone (in github.com)
- All Azure Architecture Center Hybrid and Multicloud Architectures

Monitoring

- Enable monitoring of Azure Arc enabled Kubernetes cluster
- Azure Monitor for containers overview

Application performance management

- Hybrid availability and performance monitoring

Manage data anywhere

Built-in Capabilities	Deployment Model		
	Customer Infrastructure		Azure
	SQL Server / PostgreSQL	Azure Arc enabled databases	Azure PaaS databases
Database security features	✓	✓	✓
Elastic / "Limitless" scalability	✗	◐ Limited by the capacity of customer infrastructure	✓
Automatic HA/DR	✗	◐ Customer responsible for underlying HW/K8s availability	✓
Auto upgrade, patching	✗	✓	✓
Auto backup-restore	✗	✓	✓
Monitoring	✗	✓	✓
Compliance certifications	✗ Customer responsible for compliance certification	✗ Customer responsible for compliance certification	✓ 90+ certifications
Data sovereignty	✓	✓	◐ Azure regions not available in all countries yet
Customer control	✓	✓	◐ Pre-defined HW options No control over HW/OS
Fully managed by Microsoft	✗ Customer-managed	✗ Customer-managed using software provided by Microsoft	✓
Guaranteed availability SLA	✗ Customer-managed	✗ Customer-managed using software provided by Microsoft	✓

Next steps

Performance Efficiency

Performance efficiency in a hybrid workload

Article • 11/30/2022

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. In a hybrid environment, it is important to consider how you manage your on-premises or multicloud workloads to ensure they can meet the demands for scale. You have options to scale up into the cloud when your on-premises resources reach capacity. Scale up, down, and scale out your databases without application downtime.

Using a tool like Azure Arc, you can build cloud native apps anywhere, at scale. Architect and design hybrid applications where components are distributed across public cloud services, private clouds, data centers and edge locations without sacrificing central visibility and control. Deploy and configure applications and Kubernetes clusters consistently and at scale from source control and templates. You can also bring PaaS services on premises. This allows you to use cloud innovation flexibly, where you need it by deploying Azure services anywhere. Implement cloud practices and automation to deploy faster, consistently, and at scale with always up-to-date Azure Arc enabled services. You can scale elastically based on capacity, with the ability to deploy in seconds.

Azure Arc design

The first steps with Azure Arc are to connect the machines to Azure. To use Azure Arc to connect the machine to Azure, you need to install the Azure Connected Machine agent on each machine that you plan to connect using Azure Arc. You can connect any other physical or virtual machine running Windows or Linux to Azure Arc.

There are four ways to connect machines:

1. Manual installation
2. Script-based installation
3. Connect machines at scale using service principal
4. Installation using Windows PowerShell DSC

After connecting the machines, you can then manage the VM extensions all from Azure, which provides consistent extension management between Azure and non-Azure VMs. In Azure you can use Azure Automation State Configuration to centrally store configurations and maintain the desired state of Arc enabled servers through the DSC

VM extension. You can also collect log data for analysis with Azure Monitor Logs enabled through the Log Analytics agent VM extension. With Azure Monitor, you can analyze the performance of your Windows and Linux VMs and monitor their processes and dependencies on other resources and external processes.

Azure Arc enabled Kubernetes

With Azure Arc enabled Kubernetes, you need to register the cluster first. You can register any CNCF Kubernetes cluster that is running. You'll need a kubeconfig file to access the cluster and cluster-admin role on the cluster for deploying Arc-enabled Kubernetes agents. You'll use Azure Command-Line Interface (Azure CLI) to perform cluster registration tasks.

Azure Arc enabled SQL Managed Instance

When planning for deployment of Azure Arc enabled SQL Managed Instance, you should identify the correct amount of compute, memory, and storage that will be required to run the Azure Arc data controller and the intended SQL managed instance server groups.

You have the flexibility to extend the capacity of the underlying Kubernetes or AKS cluster over time by adding additional compute nodes or storage. Kubernetes or AKS offers an abstraction layer over the underlying virtualization stack and hardware. Storage classes implement such abstraction for storage.

> ⓘ **Note**
>
> When provisioning a pod, you need to decide which storage class to use for its volumes. Your decision is important from a performance standpoint because an incorrect choice could result in suboptimal performance.

When planning for deployment of Azure Arc enabled SQL Managed Instance, you should consider a range of factors affecting storage configuration kubernetes-storage-class-factors for both data controller and database instances.

Azure Stack HCI

With the scope of Azure Arc extended to Azure Stack HCI VMs, you'll be able to automate their configuration by using Azure VM extensions and evaluate their compliance with industry regulations and corporate standards by using Azure Policy.

In remote office/branch office scenarios, you must consider storage resiliency versus usage efficiency, versus performance. Planning for Azure Stack HCI volumes involves identifying the optimal balance between resiliency, usage efficiency, and performance. The challenge results from the fact that maximizing one of these characteristics typically has a negative impact on at least one of the other two.

To learn more, see Use Azure Stack HCI switchless interconnect and lightweight quorum for Remote Office/Branch Office.

Monitoring in a hybrid environment

Monitoring in a hybrid environment can be a challenge. However, with tools like Azure Arc as you bring Azure services on-premises, you can easily enroll in additional Azure services such as monitoring, security, and update by simply turning them on.

- Across products: Integrate with Microsoft Sentinel, Microsoft Defender for Cloud
- Bring Microsoft Defender for Cloud to your on-prem data and servers with Arc
- Set security policies, resource boundaries, and RBAC for workloads across the hybrid infra
- Proper admin roles for read, modify, re-onboard, and delete a machine

Monitoring containers

Monitoring your containers is critical. Azure Monitor for containers provides a rich monitoring experience for the AKS and AKS engine clusters.

Configure Azure Monitor for containers to monitor Azure Arc enabled Kubernetes clusters hosted outside of Azure. This helps achieve comprehensive monitoring of your Kubernetes clusters across Azure, on-premises, and third-party cloud environments.

Azure Monitor for containers can provide you with performance visibility by collecting memory and processor metrics from controllers, nodes, and containers available in Kubernetes through the Metrics application programming interface (API). Container logs are also collected. After you enable monitoring from Kubernetes clusters, metrics and logs are automatically collected for you through a containerized version of the Log Analytics agent. Metrics are written to the metrics store and log data is written to the logs store associated with your Log Analytics workspace. For more information about Azure Monitor for containers, refer to Azure Monitor for containers overview.

Enable Azure Monitor for containers for one or more existing deployments of Kubernetes by using either a PowerShell or a Bash script. To enable monitoring for Arc enabled Kubernetes clusters, refer to Enable monitoring of Azure Arc enabled Kubernetes cluster.

Automatically enroll in additional Azure Arc enabled resources and services. Simply turn them on when needed:

- Strengthen your security posture and protect against threats by turning on Microsoft Defender for Cloud.
- Get actionable alerts from Azure Monitor.
- Detect, investigate, and mitigate security incidents with the power of a cloud-native SIEM, by turning on Microsoft Sentinel.

Deploy and manage containerized applications

Deploy and manage containerized applications with GitHub and Azure Policy. Ensure that applications and clusters are consistently deployed and configured at scale from source control. Write with your familiar tool set to the same application service APIs that can run consistently on-premises, across multicloud, and in edge environments. Easily instrument Azure monitoring, telemetry, and security services into your hybrid apps wherever they run.

Next steps

Reliability

Reliability in a hybrid workload

Article • 11/30/2022

In the cloud, we acknowledge up front that failures will happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. While historically you may have purchased levels of redundant higher-end hardware to minimize the chance of an entire application platform failing, in the cloud, we acknowledge up front that failures will happen.

For hybrid scenarios, Azure offers an end-to-end backup and disaster recovery solution that's simple, secure, scalable, and cost-effective, and can be integrated with on-premises data protection solutions. In the case of service disruption or accidental deletion or corruption of data, recover your business services in a timely and orchestrated manner.

Many customers operate a second datacenter, however, Azure can help reduce the costs of deploying, monitoring, patching, and scaling on-premises disaster recovery infrastructure, without the need to manage backup resources or build a secondary datacenter.

Extend your current backup solution to Azure, or easily configure our application-aware replication and application-consistent backup that scales based on your business needs. The centralized management interface for Azure Backup and Azure Site Recovery makes it simple to define policies to natively protect, monitor, and manage enterprise workloads across hybrid and cloud. These include:

- Azure Virtual Machines
- SQL and SAP databases
- On-premises Windows servers
- VMware machines

> ⓘ **Note**
>
> By not having to build on-premises solutions or maintain a costly secondary datacenter, customers can reduce the cost of deploying, monitoring, patching, and scaling disaster recovery infrastructure by backing up their hybrid data and applications with Azure.

Backup and Recovery

Together, Azure Backup and Azure Site Recovery use the underlying power and unlimited scale of the cloud to deliver high availability with minimal maintenance or monitoring overhead. These native capabilities are available through a pay-as-you-use model that bills only for the storage that is consumed.

Using Azure Site Recovery, users can set up and manage replication, failover, and failback from a single location in the Azure portal. The Azure hybrid services tool in Windows Admin Center can also be used as a centralized hub to easily discover all the available Azure services that bring value to on-premises or hybrid environments. Windows Admin Center streamlines setup and the process of replicating virtual machines on Hyper-V servers or clusters, making it easier to bolster the resiliency of environments with Azure Site Recovery's disaster recovery service.

Azure is committed to providing the best-in-class data protection to keep your applications running. Azure Backup protects backups of on-premises and cloud-resources from ransomware attacks by isolating backup data from source data, combined with multi-factor authentication and the ability to recover maliciously or accidentally deleted backup data. With Azure Site Recovery you can fail over VMs to the cloud or between cloud data centers and secure them with network security groups.

In the case of a disruption, accidental deletion, or corruption of data, customers can rest assured that they will be able to recover their business services and data in a timely and orchestrated manner. These native capabilities support low recovery-point objective (RPO) and recovery-time objective (RTO) targets for any mission-critical workload in your organization. Azure is here to help customers pivot towards a strengthened BCDR strategy.

Availability Considerations

For Azure Arc

In most cases, the location you select when you create the installation script should be the Azure region geographically closest to your machine's location. The rest of the data will be stored within the Azure geography containing the region you specify, which might also affect your choice of region if you have data residency requirements. If an outage affects the Azure region to which your machine is connected, the outage will not affect the Arc enabled server, but management operations using Azure might not be able to complete. For resilience in the event of a regional outage, if you have multiple locations which provide a geographically-redundant service, it's best to connect the machines in each location to a different Azure region.

Ensure that Azure Arc is supported in your regions by checking supported regions. Also, ensure that services referenced in the Architecture section are supported in the region to which Azure Arc is deployed.

Azure Arc enabled data services

With Azure Arc enabled SQL Managed Instance, you can deploy individual databases in either a single or multiple pod pattern. For example, the developer or general-purpose pricing tier implements a single pod pattern, while a highly available business critical pricing tier implements a multiple pod pattern. A highly available Azure SQL managed instance uses Always On Availability Groups to replicate the data from one instance to another either synchronously or asynchronously.

With Azure Arc enabled SQL Managed Instance, planning for storage is also critical from the data resiliency standpoint. If there's a hardware failure, an incorrect choice might introduce the risk of total data loss. To avoid such risk, you should consider a range of factors affecting storage configuration kubernetes-storage-class-factors for both data controller and database instances.

Azure Arc enabled SQL Managed Instance provides automatic local backups, regardless of the connectivity mode. In the Directly Connected mode, you also have the option of leveraging Azure Backup for off-site, long-term backup retention.

Azure Stack HCI

Site-level fault domains

Each physical site of an Azure Stack HCI stretched cluster represents distinct fault domains that provide additional resiliency. A fault domain is a set of hardware components that share a single point of failure. To be fault tolerant to a particular level, you need multiple fault domains at that level.

Site awareness

Site awareness allows you to control placement of virtualized workloads by designating their preferred sites. Specifying the preferred site for a stretched cluster offers many benefits, including the ability to group workloads at the site level and to customize quorum voting options. By default, during a cold start, all virtual machines use the preferred site, although it is also possible to configure the preferred site at the cluster role or group level.

Resiliency limits

Azure Stack HCI provides multiple levels of resiliency, but because of its hyper-converged architecture, that resiliency is subject to limits imposed not only by the cluster quorum, but also by the pool quorum. You can eliminate this limit by implementing cluster sets in which you combine multiple Azure Stack HCI clusters to create an HCI platform consisting of hundreds of nodes.

Next steps

Security

Security in a hybrid workload

Article • 11/30/2022

Security is one of the most important aspects of any architecture. Particularly in hybrid and multicloud environments, an architecture built on good security practices should be resilient to attacks and provide confidentiality, integrity, and availability. To assess your workload using the tenets found in the Microsoft Azure Well-Architected Framework, see the Microsoft Azure Well-Architected Review.

Microsoft Defender for Cloud can monitor on-premises systems, Azure VMs, Azure Monitor resources, and even VMs hosted by other cloud providers. To support that functionality, the standard fee-based tier of Microsoft Defender for Cloud is needed. We recommend that you use the 30-day free trial to validate your requirements. Defender for Cloud's operational process won't interfere with your normal operational procedures. Instead, it passively monitors your deployments and provides recommendations based on the security policies you enable.

Microsoft Sentinel can help simplify data collection across different sources, including Azure, on-premises solutions, and across clouds using built-in connectors. Microsoft Sentinel works to collect data at cloud scale—across all users, devices, applications, and infrastructure, both on-premises and in multiple clouds.

Azure Architecture Center (AAC) resources

- Hybrid Security Monitoring using Microsoft Defender for Cloud and Microsoft Sentinel
- DevSecOps in Azure
- Optimize administration of SQL Server instances in on-premises and multi-cloud environments by leveraging Azure Arc
- Implement a secure hybrid network
- Securely managed web applications

Principles

Azure Arc management security capabilities

- Access unique Azure security capabilities such as Microsoft Defender for Cloud.
- Centrally manage access for resources with Role-Based Access Control.

- Centrally manage and enforce compliance and simplify audit reporting with Azure Policy.

Azure Arc enabled data services security capabilities

- Protect your data workloads with Microsoft Defender for Cloud in your environment, using the advanced threat protection and vulnerability assessment features for unmatched security.
- Set security policies, resource boundaries, and role-based access control for various data workloads seamlessly across your hybrid infrastructure.

Azure Stack HCI

- **Protection in transit**. Storage Replica offers built-in security for its replication traffic. This includes packet signing, AES-128-GCM full data encryption, support for Intel AES-NI encryption acceleration, and pre-authentication integrity man-in-the-middle attack prevention.
  - Storage Replica also utilizes Kerberos AES256 for authentication between the replicating nodes.
- **Encryption at rest**. Azure Stack HCI supports BitLocker Drive Encryption for its data volumes, thus facilitating compliance with standards such as FIPS 140-2 and HIPAA.
- **Integration with a range of Azure services that provide more security advantages**. You can integrate virtualized workloads that run on Azure Stack HCI clusters with Azure services such as Microsoft Defender for Cloud.
- **Firewall-friendly configuration**. Storage Replica traffic requires a limited number of open ports between the replicating nodes.

Design

Azure Arc enabled servers

Implement Azure Monitor

Use Azure Monitor to monitor your VMs, virtual machine scale sets, and Azure Arc machines at scale. Azure Monitor analyzes the performance and health of your Windows and Linux VMs. It also monitors their processes and dependencies on other resources and external processes. It includes support for monitoring performance and application dependencies for VMs that are hosted on-premises or in another cloud provider.

Implement Microsoft Sentinel

Use Microsoft Sentinel to deliver intelligent security analytics and threat intelligence across the enterprise. This provides a single solution for alert detection, threat visibility, proactive hunting, and threat response. Microsoft Sentinel is a scalable, cloud-native, security information event management (SIEM) and security orchestration automated response (SOAR) solution that enables several scenarios including:

- Collect data at cloud scale across all users, devices, applications, and infrastructure, both on-premises and in multiple clouds.
- Detect previously undetected threats and minimize false positives.
- Investigate threats with artificial intelligence and hunt for suspicious activities at scale.
- Respond to incidents rapidly with built-in orchestration and automation of common tasks.

Azure Stack HCI

A stretched Azure Stack HCI cluster relies on Storage Replica to perform synchronous storage replication between storage volumes hosted by the two groups of nodes in their respective physical sites. If a failure affects the availability of the primary site, the cluster automatically transitions its workloads to nodes in the surviving site to minimize potential downtime.

Monitor

- Across products: Integrate with Microsoft Sentinel and Microsoft Defender for Cloud.
- Bring Microsoft Defender for Cloud to your on-premises data and servers with Arc.
- Set security policies, resource boundaries, and RBAC for workloads across the hybrid infrastructure.
- Set the correct admin roles to read, modify, re-onboard, and delete a machine.

Overview of IoT workloads

Article • 04/27/2023

This section of the Microsoft Azure Well-Architected Framework aims to address the challenges of building IoT workloads on Azure. This article describes the IoT design areas, architecture patterns, and architecture layers in the IoT workload.

Five pillars of architectural excellence underpin the IoT workload design methodology. These pillars serve as a compass for subsequent design decisions across the design areas described in this article. The remaining articles in this series delve into how to evaluate the design areas using IoT-specific design principles in the reliability, security, cost optimization, operational excellence, and performance efficiency pillars.

> **Tip**
>
> To assess your IoT workload through the lenses of reliability, security, cost optimization, operational excellence, and performance efficiency, see the **Azure Well-Architected Review**.

What is an IoT workload?

The term *workload* refers to the collection of application resources that support a common business goal or the execution of a common business process. These goals or processes use multiple services, such as APIs and data stores. The services work together to deliver specific end-to-end functionality.

Internet of Things (IoT) is a collection of managed and platform services across edge and cloud environments that connect, monitor, and control physical assets.

An *IoT workload* therefore describes the practice of designing, building, and operating IoT solutions to help meet architectural challenges according to your requirements and constraints.

The IoT workload addresses the three components of IoT systems:

- *Things*, or the physical objects, industrial equipment, devices, and sensors that connect to the cloud persistently or intermittently.
- *Insights*, information that the things collect that humans or AI analyze and turn into actionable knowledge.

- *Actions*, the responses of people or systems to insights, which connect to business outcomes, systems, and tools.

IoT architecture patterns

Most IoT systems use either a *connected products* or *connected operations* architecture pattern. Each pattern has specific requirements and constraints in the IoT design areas.

- *Connected products* architectures focus on the *hot path*. End users manage and interact with products by using real-time applications. This pattern applies to manufacturers of smart devices for consumers and businesses in a wide range of locations and settings. Examples include smart coffee machines, smart TVs, and smart production machines. In these IoT solutions, the product builders provide connected services to the product users.

- *Connected operations* architectures focus on the *warm or cold path* with edge devices, alerts, and cloud processing. These solutions analyze data from multiple sources, gather operational insights, build machine learning models, and initiate further device and cloud actions. The connected operations pattern applies to enterprises and smart service providers that connect pre-existing machines and devices. Examples include smart factories and smart buildings. In these IoT solutions, service builders deliver smart services that provide insights and support the effectiveness and efficiency of connected environments.

To learn more about the base solution architecture for IoT workloads, see Azure IoT reference architecture and Industry specific Azure IoT reference architectures.

Well-Architected Framework pillars in your IoT workload

The Azure Well-Architected Framework consists of five pillars of architectural excellence, which you can use to improve the quality of IoT workloads. The following articles highlight how IoT-specific design principles influence decisions across IoT design areas:

- *Reliability* ensures that applications meet availability commitments. Resiliency ensures that workloads are available and can recover from failures at any scale. Reliability in your IoT workload discusses how the IoT design areas of heterogeneity, scalability, connectivity, and hybridity affect IoT reliability.

- *Security* provides confidentiality, integrity, and availability assurances against deliberate attacks and abuse of data and systems. Security in your IoT workload describes how heterogeneity and hybridity affect IoT security.

- *Cost optimization* balances business goals with budget justification to create cost-effective workloads while avoiding capital-intensive solutions. Cost optimization in your IoT workload looks at ways to reduce expenses and improve operational efficiency across IoT design areas.

- *Operational excellence* covers the processes that build and run applications in production. Operational excellence in your IoT workload discusses how heterogeneity, scalability, connectivity, and hybridity affect IoT operations.

- *Performance efficiency* is a workload's ability to scale efficiently to meet demands. Performance efficiency in your IoT workload describes how heterogeneity, scalability, connectivity, and hybridity affect IoT performance.

IoT design areas

The key IoT design areas that facilitate a good IoT solution design are:

- Heterogeneity
- Security
- Scalability
- Flexibility
- Serviceability
- Connectivity
- Hybridity

The design areas are interrelated and decisions made within one area can affect decisions across the entire design. To evaluate the design areas, use the IoT-specific design principles in the five pillars of architectural excellence. These principles help clarify considerations to ensure your IoT workload meets requirements across architecture layers.

The following sections describe the IoT design areas, and how they apply to the IoT *connected products* and *connected operations* architecture patterns.

Heterogeneity

IoT solutions must accommodate various devices, hardware, software, scenarios, environments, processing patterns, and standards. It's important to identify the necessary level of heterogeneity for each architecture layer at design time.

In connected products architectures, heterogeneity describes the varieties of machines and devices that need to be supported. Heterogeneity also describes the variety of environments where you can deploy smart product, such as networks and types of users.

In connected operations architectures, heterogeneity focuses on support for different operational technology (OT) protocols and connectivity.

Security

IoT solutions must consider security and privacy measures across all layers. Security measures include:

- Device and user identity.
- Authentication and authorization.
- Data protection for data at rest and in transit.
- Strategies for data attestation.

In connected products architectures, limited control over product use in heterogeneous and widely distributed environments affects security. According to the Microsoft Threat Modeling Tool STRIDE model, the highest risk to devices is from tampering, and the threat to services is from denial of services from hijacked devices.

In connected operations architectures, the security requirements for the deployment environment are important. Security focuses on specific OT environment requirements and deployment models, such as ISA95 and Purdue, and integration with the cloud-based IoT platform. Based on STRIDE, the highest security risks for connected operations are spoofing, tampering, information disclosure, and elevation of privilege.

Scalability

IoT solutions must be able to support *hyper-scalability*, with millions of connected devices and events ingesting large amounts of data at high frequency. IoT solutions must enable proof of concept and pilot projects that start with a few devices and events, and then scale out to hyper-scale dimensions. Considering the scalability of each architecture layer is essential to IoT solution success.

In connected products architectures, scale describes the number of devices. In most cases, each device has a limited set of data and interactions, controlled by the device builder, and scalability comes only from the number of devices deployed.

In connected operations architectures, scalability depends on the number of messages and events to process. In general, the number of machines and devices is limited, but OT machines and devices send large numbers of messages and events.

Flexibility

IoT solutions build on the principle of *composability*, which enables combining various first-party or third-party components as building blocks. A well-architected IoT solution has extension points that enable integration with existing devices, systems, and applications. A high-scale, event-driven architecture with brokered communication is part of the backbone, with loosely coupled composition of services and processing modules.

In connected products architectures, changing end-user requirements define flexibility. Solutions should allow you to easily change device behavior and end-user services in the cloud, and provide new services.

In connected operations architectures, the support for different types of devices defines flexibility. Solutions should be able to easily connect legacy and proprietary protocols.

Serviceability

IoT solutions must consider ease of maintaining and repairing components, devices, and other system elements. Early detection of potential problems is critical. Ideally, a well-architected IoT solution should correct problems automatically before serious trouble occurs. Maintenance and repair operations should cause as little downtime or disruption as possible.

In connected products architectures, the wide distribution of devices affects serviceability. The ability to monitor, manage, and update devices within end user context and control, without direct access to that environment, is limited.

In connected operations architectures, serviceability depends on the given context, controls, and procedures of the OT environment, which may include systems and protocols already available or in use.

Connectivity

IoT solutions must be able to handle extended periods of offline, low-bandwidth, or intermittent connectivity. To support connectivity, you can create metrics to track devices that don't communicate regularly.

Connected products run in uncontrolled consumer environments, so connectivity is unknown and hard to sustain. Connected products architectures must be able to support unexpected extended periods of offline and low-bandwidth connectivity.

In connected operations architectures, the deployment model of the OT environment affects connectivity. Typically, the degree of connectivity, including intermittent connectivity, is known and managed in OT scenarios.

Hybridity

IoT solutions must address hybrid complexity, running on different hardware and platforms across on-premises, edge, and multicloud environments. It's critical to manage disparate IoT workload architectures, ensure uncompromised security, and enable developer agility.

In connected products architectures, the wide distribution of devices defines hybridity. The IoT solution builder controls the hardware and runtime platform, and hybridity focuses on the diversity of the deployment environments.

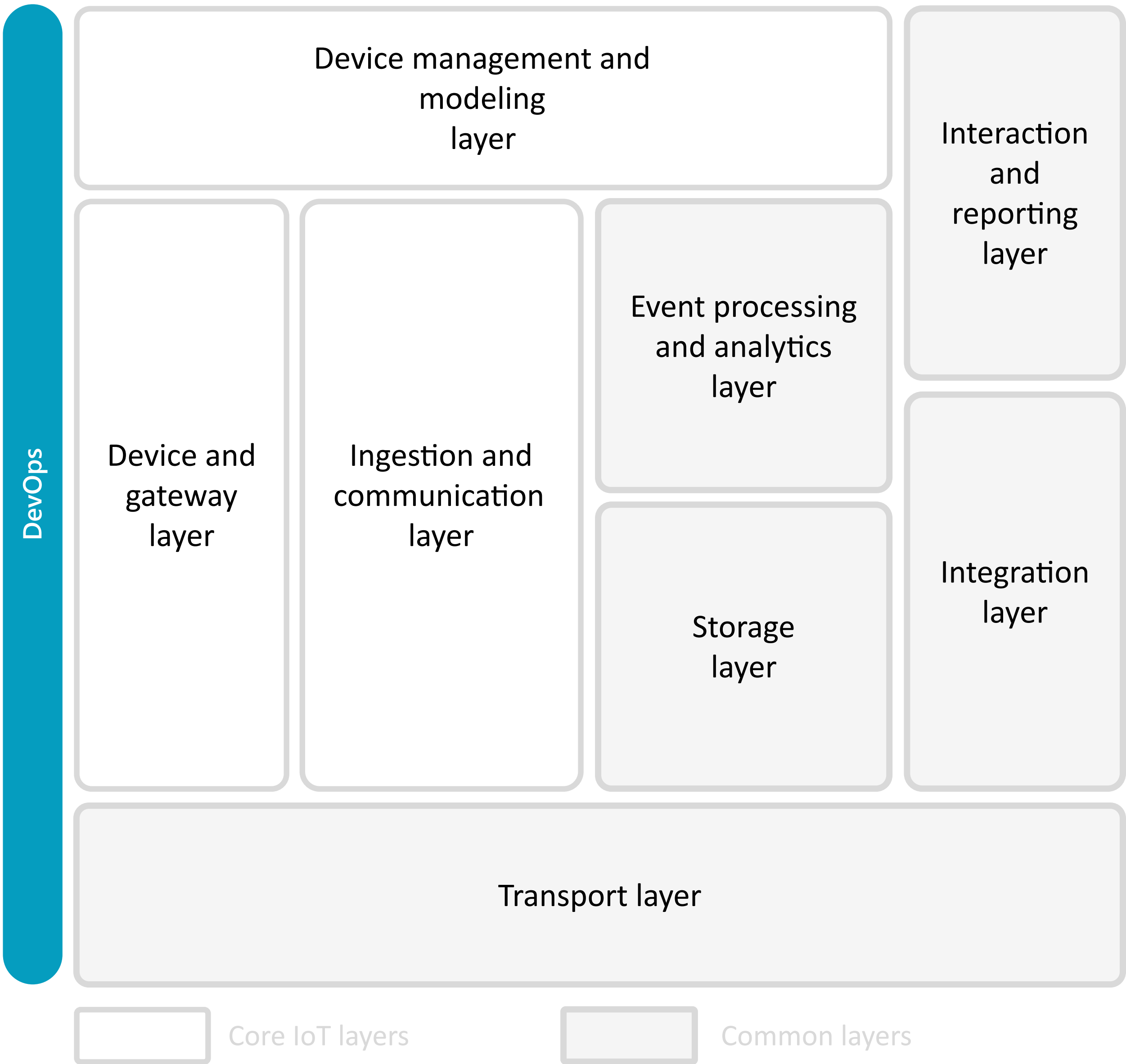
In connected operations architectures, hybridity describes the data distribution and processing logic. Scale and latency requirements determine where to process data and how fast feedback must be.

IoT architecture layers

An IoT architecture consists of a set of foundational layers. Specific technologies support the different layers, and the IoT workload highlights options for designing and creating each layer.

- *Core layers* identify IoT-specific solutions.
- *Common layers* aren't specific to IoT workloads.
- *Cross-cutting layers* support all layers in designing, building, and running solutions.

The IoT workload addresses different layer-specific requirements and implementations. The framework focuses on the *core layers*, and identifies the specific impact of the IoT workload on the *common layers*.



The following sections describe the IoT architecture layers and the Microsoft technologies that support them.

Core layers and services

The IoT core layers and services identify whether a solution is an IoT solution. The *core layers* of an IoT workload are:

- Device and gateway
- Device management and modeling
- Ingestion and communication

The IoT workload focuses primarily on these layers. To realize these layers, Microsoft provides IoT technologies and services such as:

- Azure IoT Hub
- Azure IoT device SDKs

- Azure IoT Edge
- IoT Hub Device Provisioning Service (DPS)
- Azure Digital Twins
- Azure Sphere.

> **Tip**
>
> **Azure IoT Central** is a managed application platform that you can use to quickly evaluate your IoT scenario and assess the opportunities for your business. After you've used IoT Central to evaluate your IoT scenario, you can then build your enterprise ready solution by using the power of Azure IoT platform.

Device and gateway layer

This layer represents the physical or virtual device and gateway hardware deployed at the edge or on premises. Elements in this layer include the operating systems and the device or gateway firmware. Operating systems manage the processes on the devices and gateways. Firmware is the software and instructions programmed onto devices and gateways. This layer is responsible for:

- Sensing and acting on other peripheral devices and sensors.
- Processing and transferring IoT data.
- Communicating with the IoT cloud platform.
- Base level device security, encryption, and trust root.
- Device level software and processing management.

Common use cases include reading sensor values from a device, processing and transferring data to the cloud, and enabling local communication.

Relevant Microsoft technologies include:

- Azure IoT Edge
- Azure IoT device SDKs
- Azure RTOS
- Microsoft Defender for IoT
- Azure Sphere
- Windows for IoT

Ingestion and communication layer

This layer aggregates and brokers communications between the device and gateway layer and the IoT cloud solution. This layer enables:

- Support for bi-directional communication with devices and gateways.
- Aggregating and combining communications from different devices and gateways.
- Routing communications to a specific device, gateway, or service.
- Bridging and transforming between different protocols. For example, mediate cloud or edge services into an MQTT message going to a device or gateway.

Relevant Microsoft technologies include:

- Azure IoT Hub
- Azure IoT Central

Device management and modeling layer

This layer maintains the list of devices and gateway identities, their state, and their capabilities. This layer also enables the creation of device type models and relationships between devices.

Relevant Microsoft technologies include:

- IoT Hub device twins
- IoT Hub Device Provisioning Service
- Azure Digital Twins
- IoT Plug and Play

Common layers and services

Workloads other than IoT, such as Data & AI and modern applications, also use the common layers. The top-level Azure Well-Architected Framework addresses the generic elements of these common layers, and other workload frameworks address other requirements. The following sections touch on the IoT-related influence on requirements, and include links to other guidance.

Transport layer

This layer represents the way devices, gateways, and services connect and communicate, the protocols they use, and how they move or route events, both on premises and in the cloud.

Relevant Microsoft technologies include:

- OT and IoT protocols, such as MQTT(S), AMQP(S), HTTPS, OPC-UA, and Modbus
- IoT Hub routing
- IoT Edge routes

Event processing and analytics layer

This layer processes and acts on the IoT events from the ingestion and communication layer.

- *Hot path* stream processing and analytics happen in near real-time to identify immediate insights and actions. For example, stream processing generates alerts when temperatures rise.
- *Warm path* processing and analytics identify short-term insights and actions. For example, analytics predict a trend of rising temperatures.
- *Cold path* processing and analytics create intelligent data models for the hot or warm paths to use.

Relevant Microsoft technologies include:

- Azure Stream Analytics
- Azure Functions
- Azure Databricks
- Azure Machine Learning
- Azure Synapse Analytics

Storage layer

This layer persists IoT device event and state data for some period of time. The type of storage depends on the required use for the data.

- *Streaming storage*, such as message queues, decouple IoT services and communication availability.
- *Time series-based storage* enables warm-path analysis.
- *Long-term storage* supports machine learning and AI model creation.

Relevant Microsoft technologies include:

- Azure Event Hubs
- Azure Data Explorer
- Azure Cosmos DB
- Azure SQL
- Azure Data Lake Storage

Interaction and reporting layer

This layer lets end users interact with the IoT platform and have a role-based view into device state, analytics, and event processing.

Relevant Microsoft technologies include:

- Azure App Service
- Power Apps
- Power BI
- Dynamics 365 Connected Field Service

Integration layer

This layer enables interaction with systems outside the IoT solution by using machine-to-machine or service-to-service communications APIs.

Relevant Microsoft technologies include:

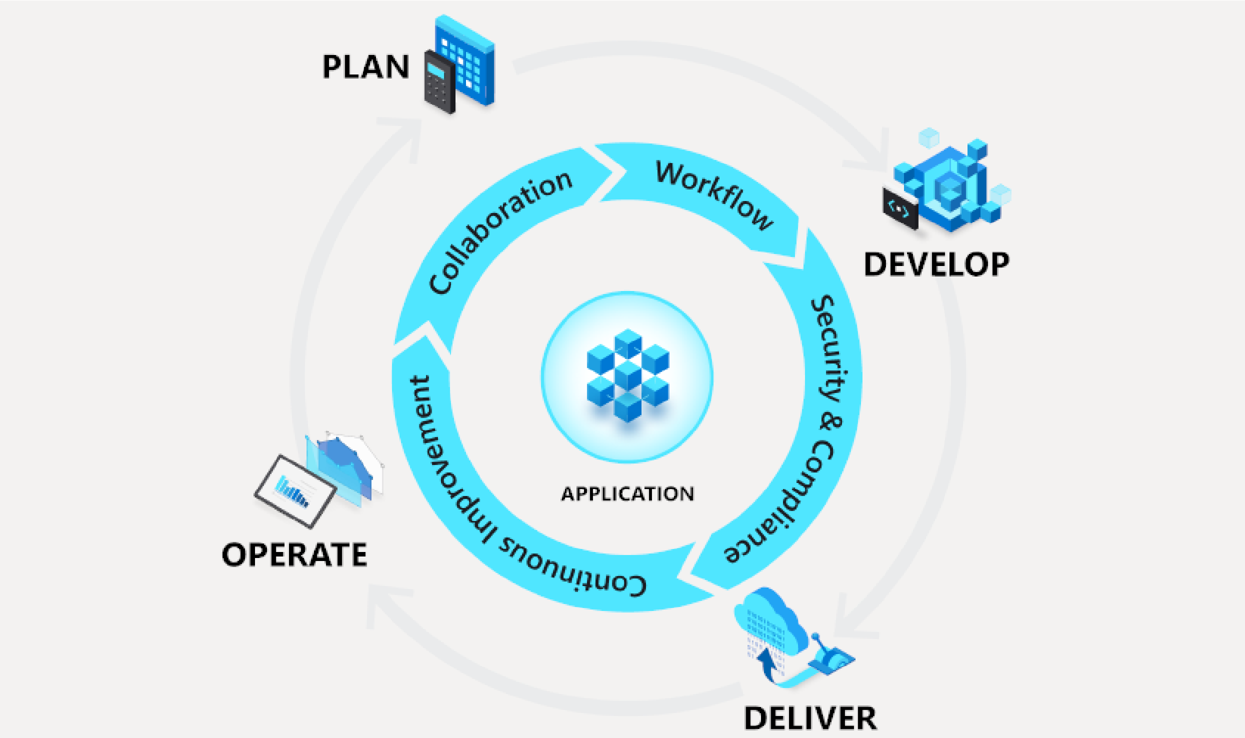
- Azure Logic Apps
- Azure Functions
- Azure API Management
- Azure Event Grid
- Power Automate

Cross-cutting activities

Cross-cutting activities like DevOps help you design, build, deploy, and monitor IoT solutions. DevOps lets formerly siloed roles, like development, operations, quality engineering, and security, coordinate and collaborate to produce better, more reliable, and agile products.

DevOps is well-known in software development, but can apply to any product or process development and operations. Teams who adopt a DevOps culture, practices, and tools can better respond to customer needs, increase confidence in the applications and products they build, and achieve business goals faster.

The following diagram shows the DevOps continuous planning, development, delivery, and operations cycle:



- Development and deployment activities include the design, build, test, and deployment of the IoT solution and its components. The activity covers all layers and includes hardware, firmware, services, and reports.
- Management and operations activities identify the current health state of the IoT system across all layers.

Correctly executing DevOps and other cross-cutting activities can determine your success in creating and running a well-architected IoT solution. Cross-cutting activities help you meet the requirements set at design time and adjust for changing requirements over time. It's important to clearly assess your expertise in these activities and take measures to ensure execution at the required quality level.

Relevant Microsoft technologies include:

- Visual Studio
- Azure DevOps
- Microsoft Security Development Lifecycle (SDL)
- Azure Monitor
- Azure Arc
- Microsoft Defender for IoT
- Microsoft Sentinel

Next steps

Reliability in your IoT workload

Security in your IoT workload

Cost optimization in your IoT workload

Operational excellence in your IoT workload

Performance efficiency in your IoT workload

Related resources

- Azure IoT reference architecture
- Azure IoT documentation

Reliability in your IoT workload

Article • 04/27/2023

IoT workloads, like all workloads, have the potential to malfunction. The key reliability considerations for well-architected IoT solutions are how quickly you can detect changes and how quickly you can resume operations.

IoT applications are often distributed at massive scale, and might operate over unreliable networks without persistent access or visibility into end-to-end data flows. Because of these factors, you should design your IoT architecture with availability and resiliency in mind.

Building a reliable IoT solution requires careful consideration of devices, cloud services, and how they interact. The choices you make for device hardware, connectivity and protocols, and cloud services affect your solution's reliability requirements and capabilities.

Assess reliability in your IoT workload

To assess your IoT workload through the lenses of the Well-Architected Framework Reliability pillar, complete the reliability questions for IoT workloads in the Azure Well-Architected Review. After the assessment identifies key reliability recommendations for your IoT solution, use the following content to help implement the recommendations.

Design Principles

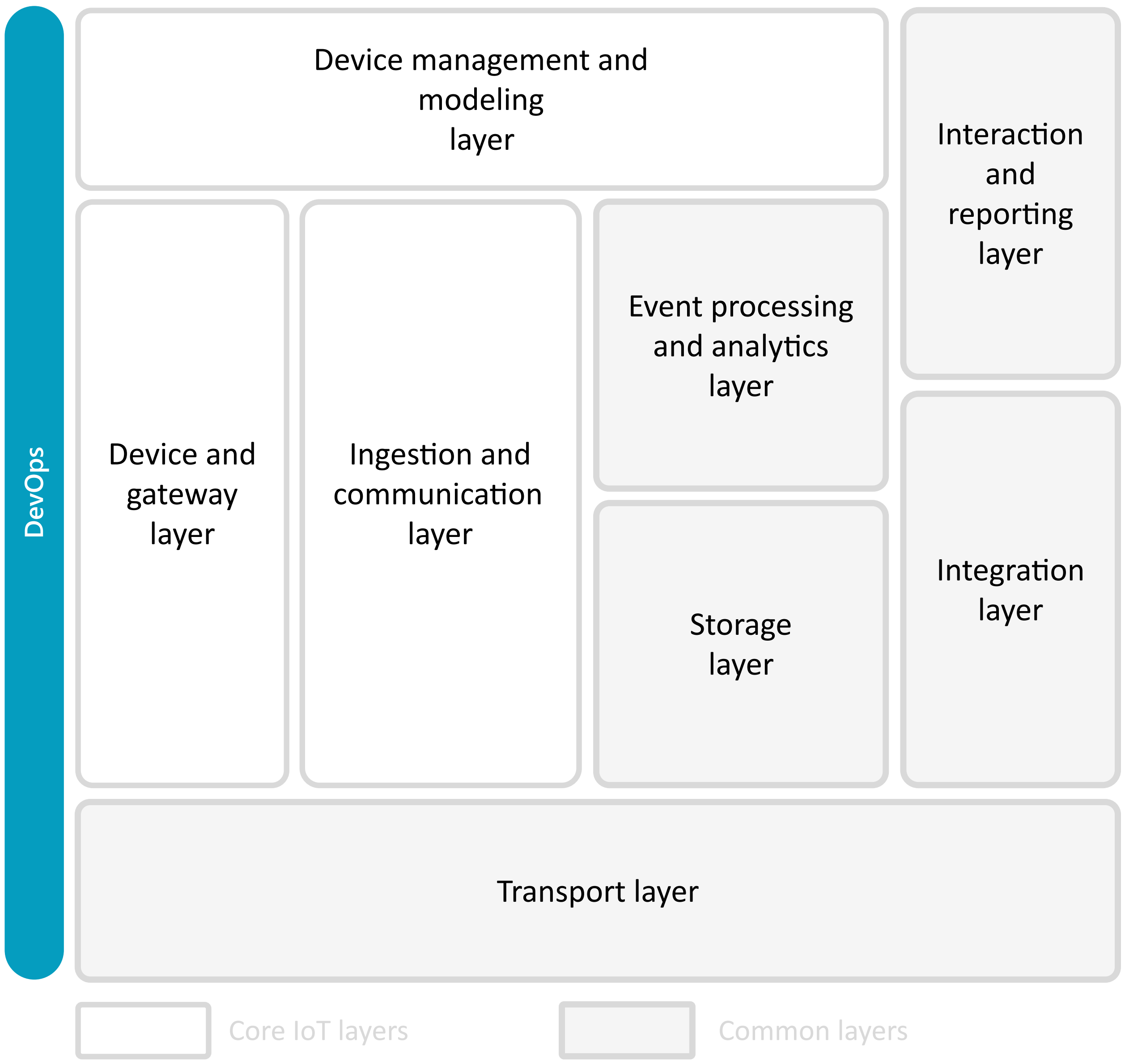
Five pillars of architectural excellence underpin the IoT workload design methodology. These pillars serve as a compass for subsequent design decisions across the key IoT design areas. The following design principles extend the quality pillar of the Azure Well-Architected Framework - Reliability.

Design principle	Considerations
Design devices for resiliency	Design your devices to satisfy the uptime and availability requirements of your end-to-end solution. Ensure that your IoT device can operate efficiently with intermittent connectivity to the cloud.

Design principle	Considerations
Design for business requirements	Cost implications are inevitable when introducing architectural modifications to meet service-level agreements (SLAs). For example, to have greater reliability and high availability you can implement cross-region redundancies and an automated system to autoscale. This trade-off should be carefully considered.
Safe, simple update procedures	An enterprise IoT solution should provide a strategy for how operators manage devices. IoT operators require simple and reliable update tools and practices.
Observe application health	Define service-level indicators (SLIs) and service-level objectives (SLOs) based on observability. Add processes for auditing, monitoring, and alerting beyond what cloud services provide.
High availability and disaster recovery (HA/DR) for critical components.	Resilient hardware and software components that build in redundancy, including cross-region redundancies.
Plan for capacity	Plan for service quotas and throttles, latency between the detection-action, and establish benchmarks at production scale to support uninterrupted data flow.

IoT architecture layers

Reliability design principles help clarify considerations to ensure your IoT workload meets requirements across the foundational IoT architecture layers. To achieve overall solution reliability, each layer should have acceptable levels of reliability.



Device and gateway layer

As part of your overall IoT solution, design your devices to satisfy your solution's end-to-end uptime and availability requirements. Devices and gateways come in many forms. IoT devices and gateways can do data collection, supervisory control, and edge analytics.

- Data collection connects devices to sensors or subscribes them to telemetry from downstream systems, and pushes collected data to the cloud. IoT solution design should ensure reliable device management and reliable communications from the device to the cloud.
- Devices that provide supervisory control not only collect data to send to the cloud, but also take actions based on that data. Devices send data back to the machines or environment to take supervisory actions. The reliability of supervisory control applications is critical.

Device design

Design and select IoT devices to function reliably in the expected operating conditions over their expected lifetimes. A reliable device should perform according to its hardware and software specifications, and any failure should be detected and managed through mitigation, repair, or replacement. Design devices for reliability, but also plan for failures.

Device lifecycle

Limited service lifetimes affect solution reliability. Assess the consequences of device failure on the solution, and define a device lifecycle strategy according to solution requirements.

Device failure impact assessment includes:

- Severity, such as single points of failures.
- Probability, such as mean time between failures.
- Detectability, such as failure mode and effects analysis.
- Acceptable downtime period.

The acceptable operational downtime determines the speed and extent of device maintenance. The availability or longevity of the device and part supply is an important consideration for device lifecycle.

The more modular the design, the easier it's to swap out parts of the system, especially if some parts become obsolete earlier than others. Alternative or multi-sourcing of component and module supply chains are critical for reliable solutions.

Environmental requirements

The conditions in which a device operates affect its reliability. Define your environmental requirements, and use devices with appropriate feature specifications. These specifications include parameters such as operating temperature range, humidity, ingress protection (IP) rating, electromagnetic interference (EMI) immunity, and shock and vibration immunity.

Operational profile

Performance stress affects the operational behavior of devices, and therefore their reliability. Define operational profiles that estimate behavior over device lifetime, and assess device reliability accordingly. Such profiles include operation modes, such as wireless transmission or low-power modes, and environmental conditions, such as temperature, over the device lifetime.

In normal operating conditions, the device and software should run safely within the specified operational profiles. Devices must be able to service and process all the external sensors and data processing that the solution requires. Avoid running at the limit of device capabilities.

Regulations and standards

Devices for specific industries are subject to applicable regulations and standards. Define regulations and standards, and make sure devices meet compliance and conformity requirements. Regulations include certification and marking, such as FCC or CE. Standards include industry or agency applications, such as ATEX and MIL-SPEC, and safety conformance, for example IEC 61508.

Device management and modeling layer

Cloud services provide each device with an identity, and manage devices at scale. The cloud is often the final data ingress point for all messages flowing from the devices. In IoT solutions, the cloud services must provide reliability for the IoT devices to integrate and transmit data.

Device connectivity conditions, including upstream to the cloud and downstream to local networks, should be part of IoT solution reliability design. Assess the potential effect of connectivity interruption or interference, and define a connectivity strategy accordingly.

The connectivity strategy should include robustness, for example fallback capability and disconnection management, and buffering backup to mitigate cloud dependency for critical or safety functions.

The following design, error handling, and monitoring practices relate to connectivity.

Connectivity design

An IoT solution should enable the flow of information between intermittently connected devices and cloud-based services. Make sure your IoT devices can operate efficiently with intermittent connectivity to the cloud.

Best practices include the following recommendations:

- Implement retry and backoff logic in device software.
- Synchronize device state with the cloud.
- Make sure you can store data on devices if your solution can't tolerate data loss.
- Use data sampling and simulations to measure network capacity and storage requirement baselines.

Connectivity implementation

The Azure IoT device SDKs provide client libraries that you can use on devices or gateways to simplify connectivity with Azure IoT services. You can use the SDKs to instrument IoT device clients that:

- Connect to the cloud.
- Provide a consistent client development experience across different platforms.
- Simplify common connectivity tasks by abstracting details of the underlying protocols and message processing patterns, such as exponential backoff with jitter and retry logic.

Connectivity monitoring

Connectivity issues for IoT devices can be difficult to troubleshoot because of the many possible points of failure. Application logic, physical networks, protocols, hardware, Azure IoT Hub, and other cloud services can have problems.

The ability to detect and pinpoint the source of an issue is critical. However, an IoT solution at scale could have thousands of devices, so it's not practical to manually check individual devices. Azure Monitor and Azure Event Grid can help you diagnose connectivity issues in IoT Hub.

Connectivity resources

- Manage connectivity and reliable messaging by using Azure IoT Hub device SDKs
- Monitor, diagnose, and troubleshoot Azure IoT Hub device connectivity
- Transient fault handling
- Error handling for resilient applications in Azure
- Circuit Breaker pattern - Cloud Design Patterns
- Compensating Transaction pattern - Cloud Design Patterns
- Throttling pattern - Cloud Design Patterns

Ingestion and communication layer

The IoT ingestion and communication layer covers service quotas and limits, capacity, throttling, and autoscale.

Redundant capacity design

When planning thresholds and alerts, consider the latency between detection and action taken. Make sure the system and operators have enough time to respond to change requests. Otherwise, for example, you might detect a need to increase the number of units, but the system could fail by losing messages before the increase can take effect.

Service quota planning

As with all platform services, IoT Hub and the IoT Hub Device Provisioning Service (DPS) enforce quotas and throttles on certain operations, so Azure can deliver predictable service levels and costs for its services. Quotas and throttles are tied to the service tier and number of units you deploy, so you can design your solution with the right number of resources. Review quotas and throttles in advance, and design your IoT Hub and DPS resources accordingly.

Production-scale benchmarks

As the number of devices or volume of data increase, the cloud gateway must scale to support uninterrupted data flow. Due to the distributed nature of IoT solutions, the number of devices, and the volume of data, it's important to establish scale benchmarks for the overall solution. These benchmarks help to plan for capacity risks. Use the Azure IoT Device Telemetry Simulator to simulate production scale volumes.

Autoscaling to dynamically adjust to quotas

A benefit of using platform as a service (PaaS) components is the ability to scale up and down with little effort according to your needs. To provide the lowest cost and operational effort, consider implementing an automated system to scale your resources up and down with the varying needs of your solution.

Quota and throttle management

To ensure solution reliability, continuously monitor resource usage against quotas and throttles to detect usage increases that indicate the need to scale. Depending on your business requirements, you can continuously monitor resource usage and alert the operator when thresholds are met, or implement an automated system to autoscale.

Capacity and scaling resources

- Understand Azure IoT Hub quotas and throttling
- Quotas and limits in the Azure IoT Hub Device Provisioning Service
- Auto-scale your Azure IoT Hub - Code Samples
- Azure IoT Device Telemetry Simulator - Code Samples
- IoT Hub IP addresses
- Best practices for device configuration within an IoT solution
- Choose the right IoT Hub tier for your solution
- Azure IoT Central quotas and limits
- IoT Hub quotas and throttling: Operation throttles
- IoT Hub quotas and throttling: Other limits

Transport layer

To connect to the cloud service for data, control, and management, devices need access to a network. Depending on the type of IoT solution, connectivity reliability might be your responsibility or that of the network service provider. Networks might have intermittent connectivity issues, and devices need to manage their behavior accordingly.

DevOps layer

An enterprise IoT solution should provide a strategy for operators to manage the system. To address reliability, IoT management and operations should use DevOps processes to handle updates, observability and monitoring, and HA/DR implementation.

Updates

The device aspect of IoT solutions presents challenges compared to cloud-based solutions. For example, there must be a way to continually update devices to address vulnerabilities and application changes.

Due to the distributed nature of IoT solutions, it's important to adopt safe and secure policies for deploying updates. IoT operators require simple and reliable update tools and practices.

A device update solution must support:

- Gradual update rollout through device grouping and scheduling controls.
- Support for resilient A/B device updates for seamless rollback.
- Detailed update management and reporting tools.

- Network optimization based on available bandwidth.

Device Update for IoT Hub is a service that enables safe, secure, and reliable over-the-air (OTA) IoT device updates. Device Update for IoT Hub can group devices and specify which devices should receive an update. Operators can view the status of update deployments and whether each device successfully applies the required updates.

If an update fails, Device Update helps operators identify the devices that failed and see the failure details. The ability to identify which devices failed can eliminate hours of trying to manually pinpoint the failure source.

Device Update monitors the status of device deployments and updates, and reports how many devices are compliant with the highest version available compatible update.

Observability and monitoring

To manage overall solution reliability and define alert procedures, you should monitor every component of your IoT solution. All Azure IoT services publish metrics that describe service health and availability. To establish end-to-end observability, also consider the metrics that you need on the device side. Use these metrics as part of your overall solution reliability monitoring.

IoT application monitoring and diagnostics are crucial for availability and resiliency. If something fails, you need to know that it failed, when it failed, and why. By monitoring the operation of an IoT application and devices against a healthy state, you can detect and fix reliability issues.

To mitigate issues that affect IoT application reliability, you must be able to capture logs and signals that help you detect issues in end-to-end operations. Use logging and monitoring to determine whether an IoT solution is functioning as expected and to help troubleshoot issues with solution components.

The following actions support observability for IoT solutions:

- Establish a mechanism to collect and analyze performance metrics and alerts.
- Configure devices, cloud services, and applications to collect and connect with Azure Monitor.
- Use real-time dashboards and alerts to monitor Azure backend services.
- Define roles and responsibilities for monitoring and acting on events and alerts. For more information, see Roles, responsibilities, and permissions.
- Implement continuous monitoring.

Azure Monitor

Azure Monitor is the recommended monitoring and visualization platform for Azure IoT solutions. You can configure devices, cloud services, and applications, regardless of deployment location, to push log messages directly or through built-in connectors into Azure Monitor.

- Use Azure Monitor built-in metrics integration for remote monitoring of IoT Edge devices. To enable this capability on your devices, add the IoT Edge Metrics Collector module to your deployment and configure it to collect and transport module metrics to Azure Monitor.

- With Azure Monitor, you can monitor the state of your IoT Hub environment, ensure it's running properly, and verify that your devices aren't being throttled or experiencing connection issues. IoT Hub provides usage metrics, such as the number of messages used and the number of devices connected. You can relay this data to Azure Monitor for analysis and to alert other services.

- If your solution uses Azure IoT Central, you can use the metrics IoT Central provides to assess the health of connected devices and active data exports. IoT Central applications enable metrics by default, which you can access from the Azure portal. Azure Monitor exposes and provides several ways to interact with these metrics.

- Azure Monitor provides custom log parsing to facilitate the decomposition of events and records into individual fields for indexing and search.

- Implement real-time dashboards and Azure Monitor alerts to monitor Azure backend services. Alerts proactively notify you about specific conditions in your monitoring data, so you can identify and address issues before customers encounter them. You can set alerts on metrics, logs, and the activity log.

Application Insights is a feature of Azure Monitor that provides extensible application performance management and monitoring for live web apps. If your IoT solution uses custom Azure App Service, Azure Kubernetes Service, or Azure Functions applications, you can use Application Insights for app monitoring and analysis.

Application Insights can:

- Automatically detect performance anomalies.
- Help diagnose issues by using powerful analytics tools.
- Show what users actually do with your apps.
- Help you continuously improve app performance and usability.

Continuous monitoring

Continuous integration and continuous deployment (CI/CD) is a DevOps practice that delivers software more quickly and reliably to provide continuous value to users. *Continuous monitoring (CM)* is a similar concept that incorporates monitoring across all phases and components of a DevOps cycle.

CM continuously ensures the health, performance, and reliability of your apps and infrastructure as they flow through development, production, and release to customers. For more information, see:

- Seven best practices for continuous monitoring with Azure Monitor
- Continuous integration and continuous deployment to Azure IoT Edge devices

Monitoring resources

- Monitor Azure IoT Hub
- Monitor Azure IoT Hub data reference
- Trace Azure IoT device-to-cloud messages with distributed tracing
- Check Azure IoT Hub service and resource health
- Collect and transport metrics - Azure IoT Edge

HA/DR for critical components

As you design and build your IoT solution, you must meet the SLA for failure recovery across the solution stack. Your SLA should guide you regarding which critical system components need HA/DR. There are multiple approaches, from redundancy across the IoT solution stack to redundancy for specific layers. Cost is also a major consideration to weigh against the importance of meeting SLAs.

- Azure IoT services have defined uptime and availability targets. Review the SLAs for Azure IoT services that are part of your solution to see if they meet your uptime goals. For example, Azure IoT Hub has an SLA of 99.9%, which means you should plan for 1 minute and 36 seconds of potential downtime per day. The Azure IoT Hub SDK provides built-in, configurable logic to handle retries and backoff.

- Consider breaking your uptime goals into two categories: device management and data ingestion operations. For example, it might be critical for a device to successfully send data to an IoT hub, even if device management services are unavailable. For more information, see the Azure IoT Hub SDK reliability features.

- Consider using redundant hardware for sensors, power, and storage. Redundant hardware enables devices to function if a critical component isn't available. Hardware can also help with connectivity issues. For example, you can use a store and forward approach for data when connectivity isn't available. Azure IoT Edge has this feature built in.

- Devices must also be able to handle cloud outages. Azure region pairing provides an HA/DR strategy for IoT Hub that meets many SLA requirements. If region pairing isn't enough, consider implementing a secondary IoT hub. You can also use DPS to avoid hardcoded IoT Hub configurations on your devices. If your primary IoT hub goes down, DPS can assign your devices to a different hub.

- Consider implementing a heartbeat message pattern for devices you expect to be online most of the time. This pattern uses a custom IoT Hub route with Azure Stream Analytics, Azure Logic Apps, or Azure Functions to determine if a heartbeat has failed. You can use the heartbeat to define Azure Monitor alerts that take actions as needed.

HA/DR resources

- Manage connectivity and reliable messaging by using Azure IoT Hub device SDKs
- IoT Hub high availability and disaster recovery
- IoT Hub intra-region HA/DR
- IoT Hub cross-region HA/DR
- How to clone an Azure IoT hub to another region
- Testing applications for availability and resiliency

Next steps

Security in your IoT workload

Related resources

- Reliability design principles
- Azure IoT reference architecture
- Azure IoT documentation

Security in your IoT workload

Article • 04/27/2023

IoT solutions have the challenge of securing diverse and heterogeneous device-based workloads with little or no direct interaction. IoT device builders, IoT application developers, and IoT solution operators share responsibility for security during the full IoT solution lifecycle. It's important to design the solution from the start with security in mind. Understand the potential threats, and add defense in depth as you design and architect the solution.

Security planning starts with a threat model. Understanding how an attacker might be able to compromise a system helps you ensure appropriate mitigations from the start. Threat modeling offers the greatest value when you incorporate it into the design phase. As part of the threat modeling exercise, you can divide a typical IoT architecture into several components or zones: device, device gateway, cloud gateway, and services. Each zone can have its own authentication, authorization, and data requirements. You can use zones to isolate damage and restrict the impact of low-trust zones on higher-trust zones. For more information, see the Internet of Things (IoT) security architecture.

The following security guidance for IoT Workloads identifies key considerations and provides design and implementation recommendations.

Assess security in your IoT workload

To assess your IoT workload through the lenses of the Well-Architected Framework Security pillar, complete the security questions for IoT workloads in the Azure Well-Architected Review. After the assessment identifies key security recommendations for your IoT solution, use the following content to help implement the recommendations.

Design Principles

Five pillars of architectural excellence underpin the IoT workload design methodology. These pillars serve as a compass for subsequent design decisions across the key IoT design areas. The following design principles extend the quality pillar of the Azure Well-Architected Framework - Security.

Design principle	Considerations

Design principle	Considerations
Strong identity	Use a strong identity to authenticate devices and users. Have a hardware root of trust for trusted identity, register devices, issue renewable credentials, and use passwordless or multi-factor authentication (MFA). Review general Azure identity and access management considerations.
Least privilege	Automate and use **least-privileged access control** to limit impact from compromised devices or identities or unapproved workloads.
Device health	Evaluate device health to gate device access or flag devices for remediation. Check security configuration, assess vulnerabilities and insecure passwords, monitor for threats and anomalies, and build ongoing risk profiles.
Device update	Continuous updates to keep devices healthy. Use a centralized configuration and compliance management solution and a robust update mechanism to ensure devices are up-to-date and healthy.
Monitor system security, plan incident response	Proactively monitor for unauthorized or compromised devices and respond to emerging threats.

Zero-trust security model

Unauthorized access to IoT systems could lead to mass information disclosure, such as leaked factory production data, or elevation of privilege for cyber-physical systems control, such as stopping a factory production line. A zero trust security model helps limit the potential impact of users gaining unauthorized access to cloud or on-premises IoT services and data.

Instead of assuming everything behind a corporate firewall is safe, zero trust fully authenticates, authorizes, and encrypts every access request before granting access. Securing IoT solutions with zero trust starts with implementing basic identity, device, and access security practices, such as explicitly verifying users, reviewing devices on the network, and using real-time risk detection to make dynamic access decisions.

The following resources can help you implement a zero-trust IoT solution:

- The Microsoft Zero Trust Assessment analyzes the gaps in your current protection for identity, endpoints, apps, network, infrastructure, and data. Use the recommended solutions to prioritize your zero trust implementation, and move forward with guidance from the Microsoft Zero Trust Guidance Center.

- The Zero Trust Cybersecurity for the Internet of Things technical paper describes how to apply a zero-trust approach to IoT solutions based on Microsoft's environment and customer experiences.
- The NIST Zero Trust Architecture (nist.gov) document provides guidance for creating zero-trust architectures. This document provides general deployment models and use cases where zero trust could improve an enterprise's overall information technology security posture.

IoT architecture patterns

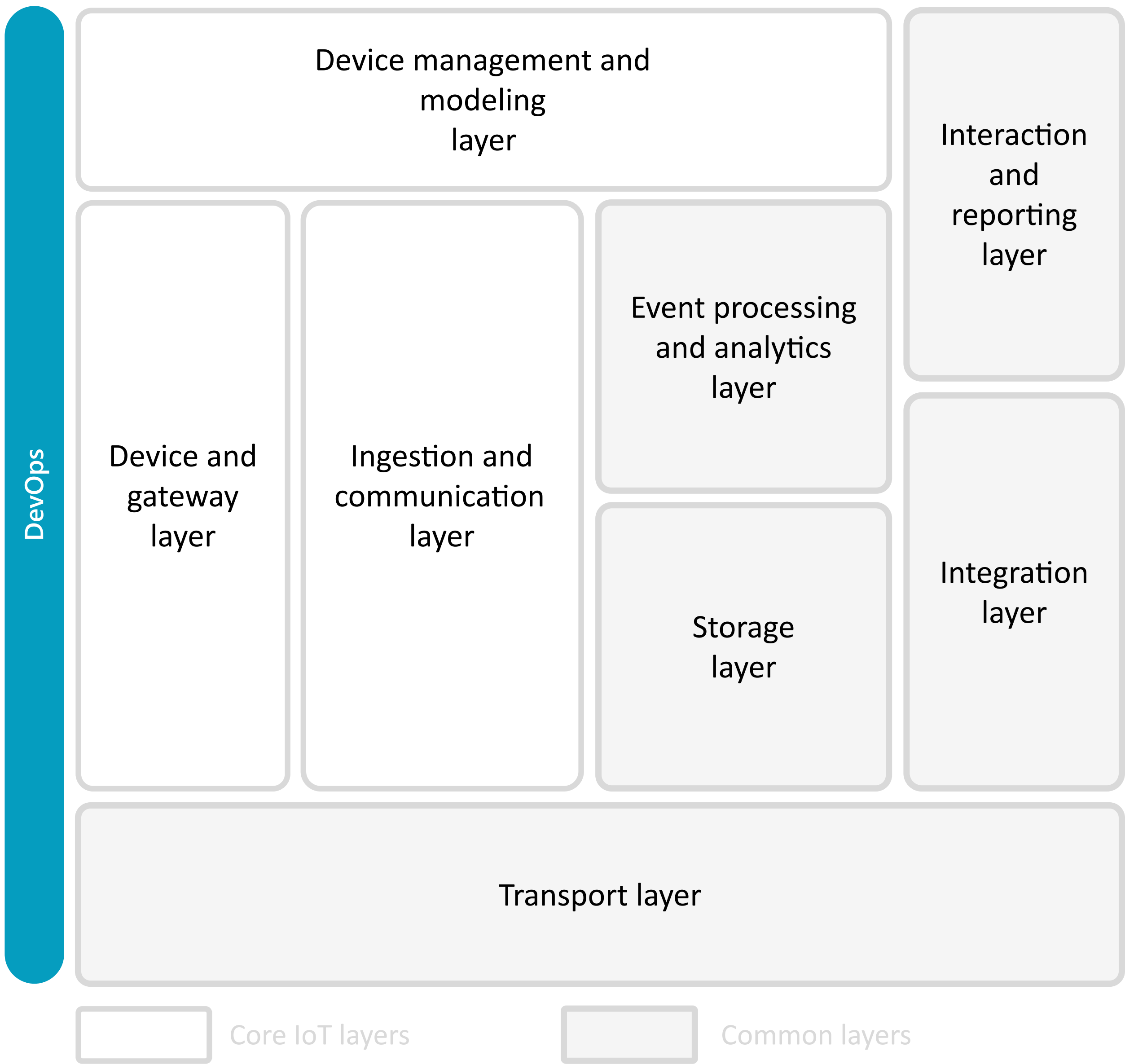
Most IoT systems use either a connected products or connected operations architecture pattern. There are key security differences between these patterns. Connected operations or *operational technology (OT)* solutions often have on-premises devices that monitor and control other physical devices. These OT devices add security challenges such as tampering, packet sniffing, and the need for out-of-band management and over-the-air (OTA) updates.

Factories and OT environments can be easy targets for malware and security breaches, because equipment can be old, physically vulnerable, and isolated from server-level security. For an end-to-end perspective, review the Azure Well-Architected Framework security pillar.

IoT architecture layers

Security design principles help clarify considerations to ensure your IoT workload meets requirements across the foundational IoT architecture layers.

All layers are subject to various threats that can be classified according to the STRIDE categories: *spoofing*, *tampering*, *repudiation*, *information disclosure*, *denial of service*, and *elevation of privilege*. Always follow Microsoft Security Development Lifecycle (SDL) practices when you design and build IoT architectures.



Device and gateway layer

This architecture layer includes the immediate physical space around the device and gateway that allows physical access or peer-to-peer digital access. Many industrial companies use the Purdue model included in the ISA 95 standard to ensure their process control networks both protect their limited network bandwidth and provide real-time deterministic behavior. The Purdue model provides an extra layer of defense-in-depth methodology.

Strong device identity

Tightly integrated capabilities of IoT devices and services provide strong device identity. These capabilities include:

- A hardware root of trust.
- Strong authentication using certificates, MFA, or passwordless authentication.

- Renewable credentials.
- Organizational IoT device registry.

A *hardware root of trust* has the following attributes:

- Secure credential storage that proves identity in dedicated, tamper-resistant hardware.
- Immutable onboarding identity tied to the physical device.
- Unique per-device renewable operational credentials for regular device access.

The *onboarding identity* represents and is inseparable from the physical device. This identity is typically created and installed during manufacturing, and can't be changed for the device's lifetime. Given its immutability and lifetime, you should use the device onboarding identity only to onboard the device into the IoT solution.

After onboarding, provision and use a renewable *operational identity* and *credentials* for authentication and authorization to the IoT application. Making this identity renewable lets you manage access and revocation of the device for operational access. You can apply policy-driven gates, such as attestation of device integrity and health, at renewal time.

The hardware root of trust also ensures that devices are built to security specifications and conform to required compliance regimes. Protect the supply chain of the hardware root of trust, or any other hardware components of an IoT device, to ensure that supply chain attacks don't compromise device integrity.

Passwordless authentication, usually using standard x509 certificates to prove a device's identity, offers greater protection than secrets such as passwords and symmetric tokens shared between both parties. *Certificates* are a strong, standardized mechanism that provides renewable passwordless authentication. To manage certificates:

- Provision operational certificates from a trusted public key infrastructure (PKI).
- Use a renewal lifetime appropriate for the business use, management overhead, and cost.
- Make renewal automatic, to minimize any potential access disruption due to manual rotation.
- Use standard, up-to-date cryptography techniques. For example, renew through certificate signing requests (CSR) instead of transmitting a private key.
- Grant access to devices based on their operational identity.
- Support credential revocation, such as a certificate revocation list (CRL) when using x509 certificates, to immediately remove device access, for example in response to compromise or theft.

Some legacy or resource-constrained IoT devices can't use a strong identity, passwordless authentication, or renewable credentials. Use IoT gateways as guardians to locally interface with these less-capable devices, bridging them to access IoT services with strong identity patterns. This practice lets you adopt zero trust today, while transitioning to use more capable devices over time.

Virtual machines (VMs), containers, or any service that embeds an IoT client can't use a hardware root of trust. Use available capabilities with these components. VMs and containers, which don't have hardware root of trust support, can use passwordless authentication and renewable credentials. A defense-in-depth solution provides redundancies where possible, and fills in gaps where necessary. For example, you could locate VMs and containers in an area with more physical security, such as a data center, compared to an IoT device in the field.

Use a centralized *organizational IoT device registry* to manage your organization's IoT device lifecycle and audit device access. This approach is similar to the way you secure the user identities of an organization's workforce to achieve zero-trust security. A cloud-based identity registry can handle the scale, management, and security of an IoT solution.

IoT device registry information onboards devices into an IoT solution by verifying that the device identity and credentials are known and authorized. After a device is onboarded, the device registry contains the core properties of the device, including its operational identity and the renewable credentials used to authenticate for everyday use.

You can use IoT device registry data to:

- View the inventory of an organization's IoT devices, including health, patch, and security state.
- Query and group devices for scaled operation, management, workload deployment, and access control.

Use network sensors to detect and inventory unmanaged IoT devices that don't connect to Azure IoT services, for awareness and monitoring.

Least-privileged access

Least-privileged access control helps limit impact from authenticated identities that might be compromised or running unapproved workloads. For IoT scenarios, grant operator, device, and workload access by using:

- Device and workload access control, for access only to scoped workloads on the device.
- Just-in-time access.
- Strong authentication mechanisms such as MFA and passwordless authentication.
- Conditional access based on a device's context, such as IP address or GPS location, system configuration, uniqueness, time of day, or network traffic patterns. Services can also use device context to conditionally deploy workloads.

To implement effective least-privileged access:

- Configure IoT cloud gateway access management to only grant appropriate access permissions for the functionality the backend requires.
- Limit access points to IoT devices and cloud applications by ensuring ports have minimum access.
- Build mechanisms to prevent and detect physical device tampering.
- Manage user access through an appropriate access control model, such as role-based or attribute-based access control.
- Layer least-privileged access for IoT devices by using network segmentation.

Network micro-segmentation

Network design and configuration provide opportunities to build defense in depth by segmenting IoT devices based on their traffic patterns and risk exposure. This segmentation minimizes the potential impact of compromised devices and adversaries pivoting to higher-value assets. Network segmentation typically uses next-generation firewalls.

Network micro-segmentation enables isolation of less-capable devices at the network layer, either behind a gateway or on a discrete network segment. Use network segmentation to group IoT devices, and use endpoint protection to mitigate the impact of potential compromise.

Implement a holistic firewall rule strategy that allows devices to access the network when required, and blocks access when not allowed. To support defense in depth, mature organizations can implement micro-segmentation policies at multiple layers of the Purdue model. If necessary, use firewalls on devices to restrict network access.

Device health

Under the zero-trust principle, device health is a key factor to determine the risk profile, including trust level, of a device. Use the risk profile as an access gate to ensure only healthy devices can access IoT applications and services, or to identify devices in questionable health for remediation.

According to industry standards, device health evaluation should include:

- Security configuration assessment and attestation that the device is configured securely.
- Vulnerability assessment to determine whether device software is out of date or has known vulnerabilities.
- Insecure credential assessment to check device credentials, such as certificates, and protocols, such as Transport Layer Security (TLS) 1.2+.
- Active threats and threat alerts.
- Anomalous behavioral alerts, such as network pattern and usage deviation.

Zero-trust criteria for devices

To support zero trust, IoT devices should:

- Contain a hardware root of trust to provide a strong device identity.
- Use renewable credentials for regular operation and access.
- Enforce least-privileged access control to local device resources such as cameras, storage, and sensors.
- Emit proper device health signals to enable enforcement of conditional access.
- Provide update agents and corresponding software updates for the usable lifetime of the device to ensure security updates can be applied.
- Include device management capabilities to enable cloud-driven device configuration and automated security response.
- Run security agents that integrate with security monitoring, detection, and response systems.
- Minimize physical attack footprint, for example by turning off or disabling any device features that aren't needed, such as physical USB or UART ports, or WiFi or Bluetooth connectivity. Use physical removal, covering, or blocking when necessary.
- Protect data on devices. If data at rest is stored on devices, use standard encryption algorithms to encrypt the data.

Several Azure products and services support IoT device security:

- Azure Sphere guardian modules connect critical legacy devices to IoT services with zero-trust capabilities, including strong identity, end-to-end encryption, and regular security updates.

- Azure IoT Edge provides an edge runtime connection to IoT Hub and other Azure services, and supports certificates as strong device identities. IoT Edge supports the PKCS#11 standard for device manufacturing identities and other secrets stored on a Trusted Platform Module (TPM) or Hardware Security Module (HSM).

- The Azure IoT Hub SDKS are a set of device client libraries, developer guides, samples, and documentation. Device SDKs implement various security features, such as encryption and authentication, to help you develop a robust and secure device application.

- Azure RTOS provides a real-time operating system as a collection of C-language libraries that you can deploy on a wide range of embedded IoT device platforms.

  Azure RTOS includes a complete TCP/IP stack with TLS 1.2 and 1.3 and basic X.509 capabilities. Azure RTOS and the Azure IoT Embedded SDK also integrate with Azure IoT Hub, Azure Device Provisioning Service (DPS), and Microsoft Defender. Features such as X.509 mutual authentication and support for modern TLS cipher suites such as ECDHE and AES-GCM cover the basics of secure network communication.

  Azure RTOS also supports:
  - Zero-trust design on microcontroller platforms that support hardware security features, such as Arm TrustZone, a memory protection and partitioning architecture.
  - Secure element devices, such as the STSAFE-A110 from ST Microelectronics.
  - Industry standards such as the Arm Platform Security Architecture (PSA), which combines hardware and firmware to provide a standardized set of security features including secure boot, cryptography, and attestation.

- The Azure Certified Device program enables device partners to easily differentiate and promote devices. The program helps solution builders and customers find IoT devices built with features that enable a zero-trust solution.

- The Edge Secured-core program (preview) validates whether devices meet security requirements for device identity, secure boot, operating system hardening, device updates, data protection, and vulnerability disclosures. The Edge Secured-core program requirements are distilled from various industry requirements and security engineering points of view.

  The Edge Secured-core program enables Azure services such as the Azure Attestation service to make conditional decisions based on device posture, thus enabling the zero-trust model. Devices must include a hardware root of trust and provide secure boot and firmware protection. These attributes can be measured by the attestation service and used by downstream services to conditionally grant access to sensitive resources.

Ingestion and communication layer

Data that's ingested into the IoT solution should be protected with the guidance in the Azure Well-Architected Framework security pillar. Additionally, for IoT solutions it's critical to ensure that communication from the device to the cloud is secure and encrypted using the latest TLS standards.

Device management and modeling layer

This architecture layer includes software components or modules running in the cloud that interface with devices and gateways for data collection and analysis, as well as for command and control.

Zero-trust criteria for IoT services

Use IoT services that offer the following key zero-trust capabilities:

- Full support for zero-trust user access control, for example strong user identities, MFA, and conditional user access.
- Integration with user access control systems for least-privileged access and conditional controls.
- A central device registry for full device inventory and device management.
- Mutual authentication, offering renewable device credentials with strong identity verification.
- Least-privileged device access control with conditional access so only devices fulfilling criteria, such as health or known location, can connect.
- OTA updates to keep devices healthy.
- Security monitoring of both IoT services and connected IoT devices.
- Monitoring and access control for all public endpoints, and authentication and authorization for any calls to these endpoints.

Several Azure IoT services provide these zero-trust capabilities.

- Windows for IoT helps ensure security across key pillars of the IoT security spectrum.
  - BitLocker Drive Encryption, Secure Boot, Windows Defender Application Control, Windows Defender Exploit Guard, secure Universal Windows Platform (UWP)

applications, Unified Write Filter, a secure communication stack, and security credential management protect data at rest, during code execution, and in transit.

  - Device Health Attestation (DHA) detects and monitors trusted devices to let you start with a trusted device and maintain trust over time.

  - Device Update Center and Windows Server Update Services apply the latest security patches. You can remediate threats to devices by using Azure IoT Hub device management features, Microsoft Intune or third-party mobile device management solutions, and Microsoft System Center Configuration Manager.

- Microsoft Defender for IoT is an agentless, network layer security platform that delivers continuous asset discovery, vulnerability management, and threat detection for IoT and OT devices. Defender for IoT continuously monitors network traffic using IoT-aware behavioral analytics to identify unauthorized or compromised components.

  Defender for IoT supports proprietary embedded OT devices and legacy Windows systems commonly found in OT environments. Defender for IoT can inventory all IoT devices, assess for vulnerabilities, provide risk-based mitigation recommendations, and continuously monitor devices for anomalous or unauthorized behavior.

- Microsoft Sentinel, a cloud-based security information and event management (SIEM) and security orchestration, automation, and response (SOAR) platform, that tightly integrates with Microsoft Defender for IoT. Microsoft Sentinel provides a cloud-scale view of security across your enterprise by collecting data across all users, devices, applications, and infrastructure, including firewalls, network access control, and network switch devices.

  Microsoft Sentinel can quickly spot anomalous behaviors that indicate potential compromise of IoT or OT devices. Microsoft Sentinel also supports third-party security operations center (SOC) solutions such as Splunk, IBM QRadar, and ServiceNow.

- Azure IoT Hub provides an operational registry for IoT devices. IoT Hub accepts device operational certificates to enable strong identity, and can disable devices centrally to prevent unauthorized connections. IoT Hub supports provisioning of module identities that support IoT Edge workloads.

  - Azure IoT Hub Device Provisioning Service (DPS) provides a central device registry for organizational devices to register for onboarding at scale. DPS accepts device certificates to enable onboarding with strong device identity and renewable credentials, registering devices in IoT Hub for their daily operation.

  - Azure Device Update (ADU) for IoT Hub lets you deploy OTA updates for your IoT devices. ADU provides a cloud-hosted solution to connect virtually any device, and supports a broad range of IoT operating systems, including Linux and Azure RTOS.

  - Azure IoT Hub support for virtual networks lets you restrict connectivity to IoT Hub through a virtual network that you operate. This network isolation prevents connectivity exposure to the public internet, and can help prevent exfiltration attacks from sensitive on-premises networks.

The following Microsoft products fully integrate hardware and Azure services in overall IoT solutions.

- Azure Sphere is a fully managed integrated hardware, OS, and cloud platform solution that helps medium and low-power IoT devices attain the seven properties of highly secured devices to implement zero trust. Devices use explicit verification and implement certificate-based Device Attestation and Authentication (DAA), which automatically renews trust.

  Azure Sphere uses least-privileged access, where applications are denied access by default to all peripheral and connectivity options. For network connectivity, permitted web domains must be included in the software manifest or the application can't connect outside of the device.

  Azure Sphere is built around assumed breach. Defense in depth layers protections throughout the OS design. A secure world partition running in Arm TrustZone on Azure Sphere devices helps segment OS breaches from access to Pluton or hardware resources.

  Azure Sphere can be a guardian module to secure other devices, including existing legacy systems not designed for trusted connectivity. In this scenario, an Azure Sphere guardian module deploys with an application and interfaces with existing devices through Ethernet, serial, or BLE. The devices don't necessarily have direct internet connectivity.

- Azure Percept is an end-to-end edge AI platform that can help you start a proof of concept in minutes. Azure Percept includes hardware accelerators integrated with Azure AI and IoT services, pre-built AI models, and solution management.

  Azure Percept devices use a hardware root of trust to help protect inference data, AI models, and privacy-sensitive sensors like cameras and microphones. Azure

Percept enables device authentication and authorization for Azure Percept Studio services. For more information, see Azure Percept security.

DevOps layer

An enterprise IoT solution should provide a strategy for operators to manage the system. DevOps methodologies that proactively focus on security include:

- Centralized configuration and compliance management, to securely apply policies and distribute and update certificates.
- Deployable updates, to update the full set of software on devices, firmware, drivers, base OS and host applications, and cloud-deployed workloads.

For more information, see Enable DevSecOps with Azure and GitHub.

Continuous updates

To control device access based on health, you must proactively maintain production devices in a working, healthy target state. Update mechanisms should:

- Have remote deployment capabilities.
- Be resilient to changes in environment, operating conditions, and authentication mechanism, such as certificate changes because of expiry or revocation.
- Support update rollout verification.
- Integrate with pervasive security monitoring to enable scheduled updates for security.

You should be able to defer updates that interfere with business continuity, but eventually complete them within a well-defined time interval after you detect a vulnerability. Devices that haven't been updated should be flagged as unhealthy.

Security monitoring and response

An IoT solution needs to be able to perform monitoring and remediation at scale for all its connected devices. As a defense-in-depth strategy, monitoring adds an extra layer of protection for managed greenfield devices, and provides a compensating control for legacy, unmanaged brownfield devices that don't support agents and can't be patched or configured remotely.

You need to decide on logging levels, types of activities to monitor, and responses required for alerts. Logs should be stored securely and not contain any security details.

According to the Cybersecurity and Infrastructure Security Agency (CISA), a security monitoring program should monitor and audit unauthorized changes to controllers, unusual behavior from devices, and access and authorization attempts. Security monitoring should include:

- Generating an as-is asset inventory and network map of all IoT and OT devices.
- Identifying all communication protocols used across IoT and OT networks.
- Cataloging all external connections to and from networks.
- Identifying vulnerabilities in IoT and OT devices and using a risk-based approach to mitigate them.
- Implementing a vigilant monitoring program with anomaly detection to detect malicious cyber tactics such as *living off the land* within IoT systems.

Most IoT attacks follow a *kill chain* pattern, where adversaries establish an initial foothold, elevate their privileges, and move laterally across the network. Often, attackers use privileged credentials to bypass barriers such as next-generation firewalls established to enforce network segmentation across subnets. Rapidly detecting and responding to these multistage attacks requires a unified view across IT, IoT, and OT networks, combined with automation, machine learning, and threat intelligence.

Collect signals from the entire environment, including all users, devices, applications, and infrastructure, both on-premises and in multiple clouds. Analyze the signals in centralized SIEM and extended detection and response (XDR) platforms, where SOC analysts can hunt for and uncover previously unknown threats.

Finally, use SOAR platforms to respond to incidents rapidly and mitigate attacks before they materially impact your organization. You can define playbooks that automatically execute when specific incidents are detected. For example, you can automatically block or quarantine compromised devices so they're unable to infect other systems.

Next steps

Cost optimization in your IoT workload

Related resources

- How to apply a Zero Trust approach to your IoT solutions
- Zero Trust Cybersecurity for the Internet of Things
- Internet of Things (IoT) security architecture
- Industry IoT Consortium Security Maturity Model
- Security design principles

- Azure IoT reference architecture
- Azure IoT documentation

Cost optimization in your IoT workload

Article • 04/27/2023

Cost effectiveness is one of the key success factors for IoT projects. In a typical IoT solution, devices generate large quantities of telemetry that they send to the cloud for cloud technologies to process and store. How you develop devices and applications, handle large volumes of data, and design your architecture affects overall costs.

Because an IoT solution is a multilayered technology stack, there are many cost-saving factors to consider, and many opportunities to optimize costs. Cost optimization is a process of closed-loop cost control that needs to be continuously monitored, analyzed, and improved throughout a solution lifecycle.

Solution requirements are the key criteria for IoT architecture decisions. You can separate requirements into functional and operational requirements. Separate the cost considerations for each type of requirement, because functional requirements determine system design, while operational requirements affect system architecture. Develop multiple use cases based on requirements and compare them before finalizing your design.

This article presents cost considerations for various combinations of Azure IoT services and technologies. For cost optimization for specific industries or use cases like connected factories, predictive maintenance, or remote monitoring, see Industry specific Azure IoT reference architectures.

Assess cost optimization in your IoT workload

To assess your IoT workload through the lenses of the Well-Architected Framework Cost Optimization pillar, complete the cost optimization questions for IoT workloads in the Azure Well-Architected Review. After the assessment identifies key cost optimization recommendations for your IoT solution, use the following content to help implement the recommendations.

Design Principles

Five pillars of architectural excellence underpin the IoT workload design methodology. These pillars serve as a compass for subsequent design decisions across the key IoT design areas. The following design principles extend the quality pillar of the Azure Well-Architected Framework - Cost Optimization.

Design principle	Considerations
Set up budgets and maintain cost constraints	Understand total cost of ownership (TCO) by accounting for both direct and indirect costs when planning.
Use industry-standard strategies and approaches	For IoT specific industries with their own ecosystems, for example manufacturing, energy and environment, or automotive and transportation, use industry-standard strategies and approaches.
Choose the correct resources	Define implementation plans for each IoT architecture layer.
Continuously monitor and optimize cost management	Monitor and optimize costs with ongoing cost optimization activities after you implement your solution.

Total cost of ownership (TCO)

IoT costs are a tradeoff between various technology options. Sometimes it's not a simple comparison, because IoT is an end-to-end system. Consider the cost benefits of synergy when reconciling multiple services and technologies. For example, you can use Azure IoT Hub device twins to handle events in Azure Digital Twins. Device Twins in IoT Hub are available only in the standard tier of IoT Hub.

It's important to correctly estimate long-term aggregated costs. Review the IoT technology stacks, and develop a cost model that includes the costs to implement and operate all services involved. The Azure Pricing Calculator helps estimate both startup and operational costs.

In some areas, a one-time cost can be more effective than recurring costs. For example, in security where hacking techniques are always changing, it can be best to import a reliable commercial operating system and module such as Azure Sphere. For a one-time payment, such services provide ongoing monthly device security patches.

Estimate solution costs based on running at scale in production, not proof-of-concept (PoC) architecture. Architecture and costs evolve rapidly after the PoC. According to the IoT Signals EDITION 3 report, the top reason for PoC failure is the high cost of scaling. The high cost of scaling IoT projects comes from the complexities of integrating across layers, such as devices, edge connectivity, and compatibility across applications.

Your cost model should include the following areas:

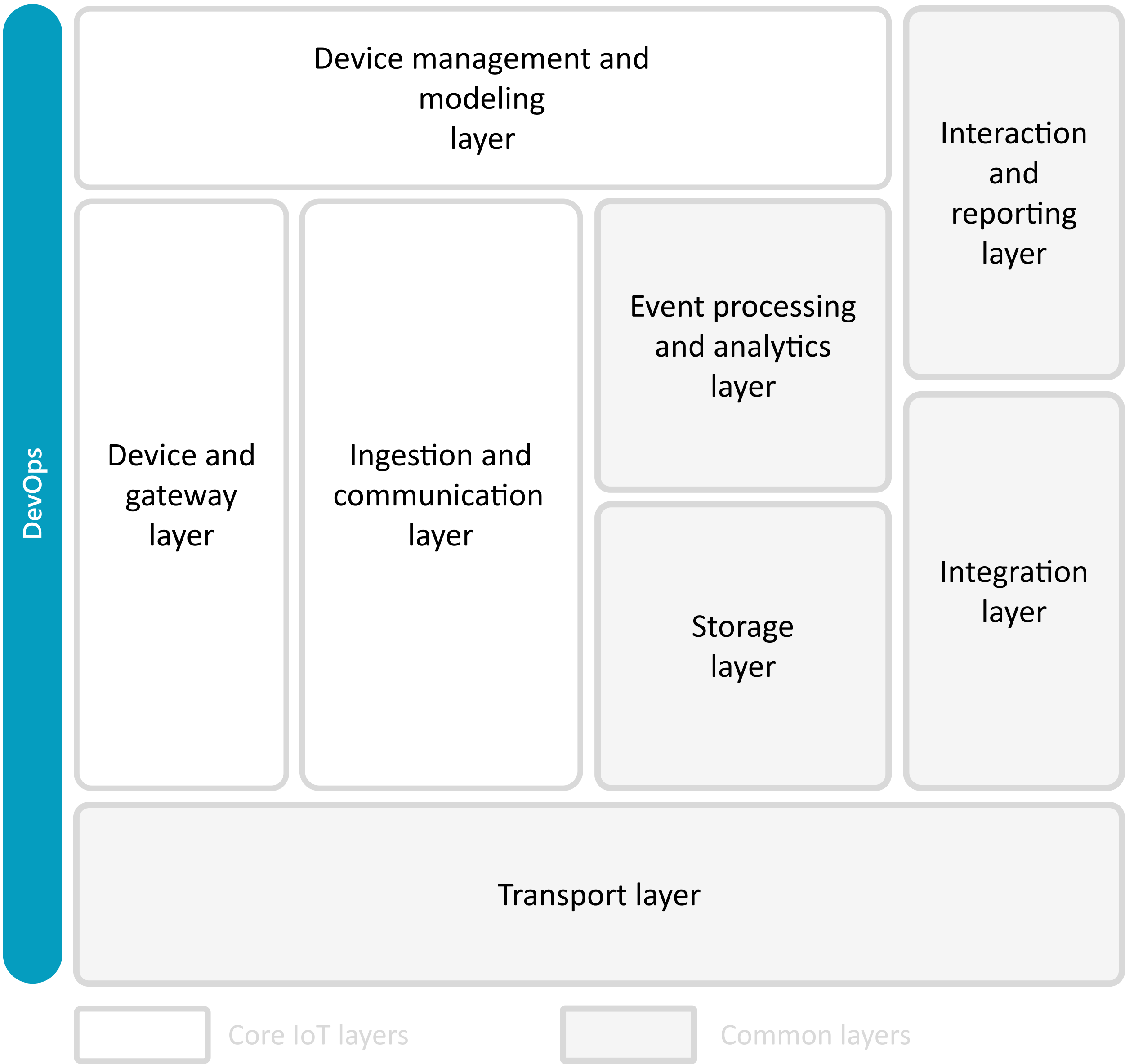
- Devices: Starting with a limited number of connected devices, estimate growth in the number of devices deployed and their messaging patterns. Both devices and messages can have linear or non-linear growth over time.
- Infrastructure: To evaluate infrastructure costs, account for the basics first: storage, compute, and network. Then account for all services your solution needs to ingest, egress, and prepare data.
- Operations: Include long-term operational costs that increase in parallel with infrastructure costs, such as employing operators, vendors, and customer support teams.
- Monitoring: Continuously monitor and review costs to identify gaps between planned and actual costs. A regular cost review meeting helps achieve cost optimization.

IoT architecture layers

Cost Optimization design principles help clarify considerations to ensure your IoT workload meets requirements across the foundational IoT architecture layers.

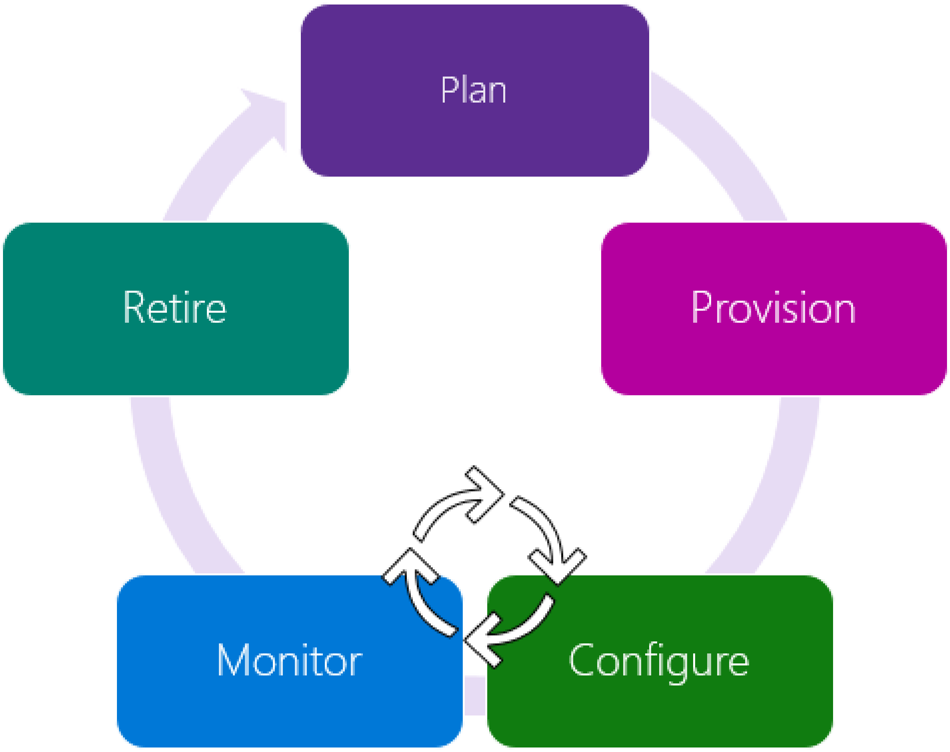
Understanding the IoT architecture layers helps you define a cost baseline and consider multiple architectures for cost comparison. Each layer has multiple technologies and ecosystems options, such as devices, telecommunications, or the edge location, so you need to establish a cost strategy for each layer.

The IoT core layers: device and gateway, device management and modeling, and ingestion and communication, identify IoT-specific solutions. The other layers and cross-cutting activities are also common to, and often shared with, other workloads. However, TCO and cost optimization must take all costs into account, so you need to consider the IoT-related costs of common and cross-cutting activities as well as the IoT-specific layers.



Device and gateway layer

This layer is responsible for generating, in some cases optimizing, and transferring data to the cloud. Cost is a key consideration for designing this layer. Cost optimization should account for the entire device lifecycle of plan, provision, configure, monitor, and retire.



Edge solutions require IoT devices to be deployed in the field. The deployment might need networking and power supply infrastructure that affects costs. Pre-existing infrastructure can minimize installation costs, but might require ensuring that the installation doesn't affect existing systems.

Developing or installing IoT devices might require training and employing dedicated internal or external personnel. Required skills include hardware design, embedded application development, cloud and local connectivity, security and privacy, and IoT solution architecture. Industry-specific expertise might also be required. Include these costs in overall device costs.

Device costs include organizing logistics, such as storage, inventory management, and transport. Include the cost of any decommissioning activities when devices reach the end of their operational lifecycle.

For devices connected to the cloud, optimize data transmissions to maintain cost boundaries. Strategies include minimizing payload sizes, batching messages, and transmitting during off-peak periods. These optimizations also incur costs to implement.

To learn more about Azure IoT devices, see:

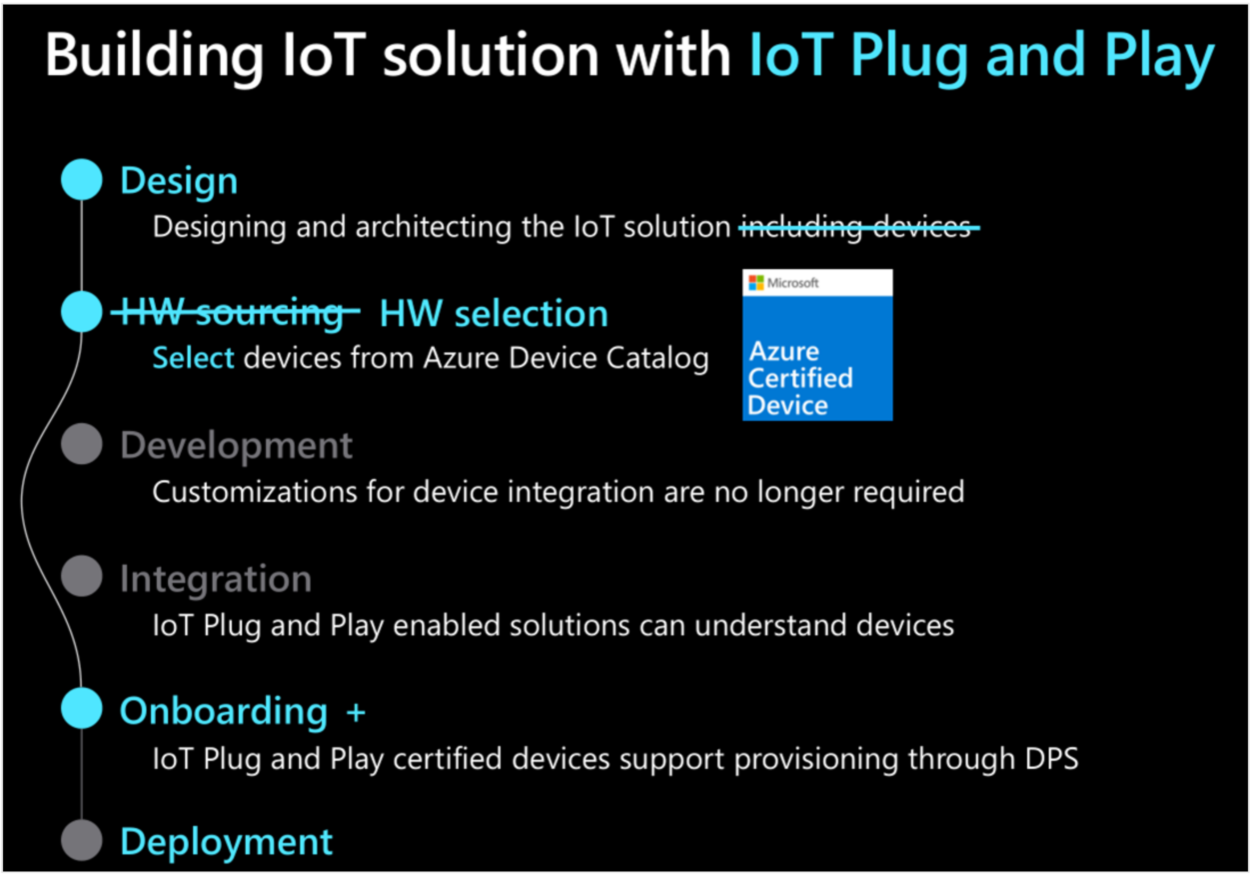
- Overview of Azure IoT device types
- Best practices for device configuration within an IoT solution

Hardware selection

Most of the device development process depends on hardware selection. A make-or-buy decision for devices takes into account qualitative factors like WiFi certification and quantitative factors like bill of materials cost and time to market. Choosing between off-the-shelf hardware or a custom design affects IoT device cost and time to market.

- Off-the-shelf devices might cost more per unit, but have predictable costs and lead times. Off-the-shelf devices also remove the need for complex supply chain management.
- Custom devices can reduce unit costs, but involve development time, and incur non-recurring engineering costs such as design, test, certification submissions, and manufacture.
- Pre-certified system components or modules can reduce time to market and create a semi-custom device, but are more expensive than discrete chips. You need to properly resource supply-chain and inventory management.

The Azure Certified Device catalog offers devices that work well with Azure IoT and can help reduce costs and time to market. You'll focus on designing and architecting the IoT solution with the flexibility to select the hardware from an extensive list of certified devices. IoT Plug and Play devices can reduce both device and cloud development costs. When you select an Azure Certified Device, you can skip device customizations and integration straight to onboarding into your IoT Solution.



Lambda architectural pattern

IoT solutions commonly use the hot/warm/cold lambda architectural pattern in the cloud. This pattern also applies to the edge when you use more performant edge devices or the Azure IoT Edge runtime. Optimizing this pattern on the edge reduces overall solution costs. You can choose the most cost-effective service for cloud data ingestion and processing.

- Hot path processing includes near real-time processing, process alerts, or edge notifications. You can use Azure IoT Hub event streams to process alerts in the cloud.
- Warm path processing includes using storage solutions on the edge, such as open-source time-series databases or Azure SQL Edge. Azure SQL Edge includes edge stream processing features and time-series optimized storage.
- Cold path processing includes batching lower importance events and using a file transfer option through the Azure Blob Storage module. This approach uses a lower cost data transfer mechanism compared to streaming through IoT Hub. After cold data arrives in Azure Blob storage, there are many options to process the data in the cloud.

Device security

Both IoT Hub with Device Provisioning Service (DPS) and IoT Central support device authentication with symmetric keys, trusted platform module (TPM) attestation, and X.509 certificates. There's a cost factor associated with each option.

- X.509 certificates are the most secure option for authenticating to Azure IoT Hub, but certificate management can be costly. Lack of certificate lifecycle management planning can make certificates even costlier. Typically, you work with third-party vendors who offer CA and certificate management solutions. This option requires using a public key infrastructure (PKI). Options include a self-managed PKI, a third-party PKI, or the Azure Sphere security service, which is available only with Azure Sphere devices.
- TPMs with X.509 certificates offer an added layer of security. DPS also supports authentication through TPM endorsement keys. The main costs are from hardware, potential board redesign, and complexity.
- Symmetric key authentication is the simplest and lowest cost option, but you must evaluate the impact on security. You need to protect keys on the device and in the

cloud, and securely storing the key on the device often requires a more secure option.

Review costs associated with each of these options, and balance potentially higher hardware or services costs with increased security. Integration with your manufacturing process can also influence overall costs.

For more information, see Security practices for Azure IoT device manufacturers.

Azure RTOS

Azure RTOS is an embedded development suite for devices. Azure RTOS includes a small but powerful operating system that provides reliable, ultra-fast performance for resource-constrained devices. Azure RTOS is easy to use and has been deployed on more than 10 billion devices. Azure RTOS supports the most popular 32-bit microcontrollers and embedded development tools, so you can make the most of existing device builder skills.

Azure RTOS is free for commercial deployment using pre-licensed hardware. Azure RTOS comes with Azure IoT cloud capabilities and features such as device update and security. These features help reduce both device and cloud development costs.

Azure RTOS is certified for safety and security, helping to reduce the time and cost of building compliant devices for specific verticals such as medical, automotive, and manufacturing.

LPWAN devices

If LPWAN devices, such as LoRaWAN, NB-IoT, or LTE-M, are already connected to another IoT cloud, the Azure IoT Central Device Bridge can help bridge to Azure IoT Central. Azure IoT Central Device Bridge lets you focus on adding industry knowledge, and evaluating the solution without incurring costs to change existing devices.

When building your enterprise ready solution, you'll need to consider the costs to integrate LPWAN devices with Azure IoT Hub.

Azure Sphere

Azure Sphere is a secure, end-to-end IoT solution platform with built-in communication and security features for internet-connected devices. Azure Sphere comprises a secured, connected, crossover microcontroller unit (MCU), a custom high-level Linux-based operating system (OS), and a cloud-based security service that provides continuous, renewable security. Azure Sphere reduces the effort to build and maintain a secure environment from device to the cloud.

Azure Sphere provides OS updates and zero-day renewable security for 10 years on top of X.509 based PKI, user app updates, error reporting, and device management beyond 10 years without extra cost. Azure Sphere reduces the operational cost of keeping millions of devices up to date with the latest security.

Azure Stack

Azure Stack solutions extend Azure services and capabilities to environments beyond Azure datacenters, such as on-premises datacenters or edge locations. Azure Stack solutions include Azure Stack Edge and Azure Stack HCI.

- Azure Stack Edge is an Azure-managed appliance that's ideal for hardware-accelerated machine learning workloads at edge locations. Azure Stack Edge runs on modern technology stacks such as containers, so Azure Stack Edge deployed in an edge location can serve multiple workloads. Sharing computational power among workloads reduces TCO.

- Azure Stack HCI is a purpose-built, hyperconverged solution with native Azure integration. Azure Stack HCI offers scalable virtualization to host IoT solutions. Virtualization brings extra benefits such as security, scalability, and flexible environments, which can reduce TCO by sharing the hardware with other workloads. Azure Stack HCI offers more compute power than Azure Stack Edge and is ideal for industry process transformation.

Azure Stack solutions bring Azure capability to the edge, but hardware sizing constrains the total compute power. Identify use cases and estimated compute power, and factor in sizing to match costs to performance needs.

Azure public or private MEC

IoT devices can generate large quantities of data, and might also have strong requirements for low power consumption and low costs. Small, inexpensive IoT devices are designed for one or a few tasks, such as collecting sensor or location data and offloading it for further processing.

Azure public or private multi-access edge compute (MEC) and 5G help optimize the costs of offloading data from devices. MEC-based IoT solutions enable low-latency data processing at the edge instead of on devices or in the cloud. Latency is 1-5 ms instead of the typical 100-150 ms for the cloud. MEC-based IoT solutions are flexible, and the devices themselves are inexpensive, operate with minimal maintenance, and use smaller, cheaper, and long-lasting batteries. MEC keeps data analytics, AI, and optimization functions at the edge, which keeps IoT solutions simple and inexpensive.

In addition to serving as an edge processing, compute, and 5G communication device for IoT workloads, MEC serves other workloads as a communication device to establish high-speed connections to the public cloud or remote sites.

Azure IoT Edge

Azure IoT Edge has built-in capabilities for high message volumes. Azure IoT Edge managed devices with gateway capabilities can reduce network costs and minimize the number of messages through local processing and edge scenarios.

Avoid device-to-device or module-to-module edge communications or device-to-cloud interactions that use many small messages. Use built-in message batching features to send multiple telemetry messages to the cloud. These features can help reduce the costs of using IoT Hub. Reducing both the number of daily messages and the number of device-to-cloud operations per second can allow choosing a lower tier in IoT Hub. To learn more, see Stretching the IoT Edge performance limits.

To reduce data exchange costs, you can deploy Azure services such as Azure Stream Analytics and Azure Functions to IoT Edge. Azure Stream Analytics and Azure Functions can aggregate and filter large volumes of data at the edge and send only important data to the cloud. Azure Blob Storage on IoT Edge can reduce the need to transfer large quantities of data over the network. Edge storage is useful for transforming and optimizing large quantities of data before sending it to the cloud.

Free Azure IoT Edge modules for open protocols such as OPC Publisher and Modbus help connect various devices with minimal development. If upload performance is critical, choosing a proven IoT Edge module from a vendor can be more cost effective than building a custom module. You can search for and download IoT Edge modules from the Azure Marketplace.

Ingestion and communication layer

A cloud IoT gateway is a bridge between devices and cloud services. As a front-end service to the cloud platform, a gateway can aggregate all data with protocol translation and provide bi-directional communication with devices.

There are many factors to take into account for device to IoT gateway communications, such as device connectivity, network, and protocol. An understanding of IoT communication protocols, network types, and messaging patterns helps you design and optimize a cost effective architecture.

For device connectivity, it's important to specify the network type. If you select a private LAN or WAN solution, such as WiFi or LoraWAN, consider network TCO as part of overall costs. If you use carrier networks such as 4G, 5G, or LPWAN, include recurring connectivity costs.

IoT solution platform

To build an IoT solution for your business, you typically evaluate your solution by using the managed app platform approach and build your enterprise ready solution by using the platform services.

- Platform services let you fine-tune services and control overall costs. It provides all the building blocks for customized and flexible IoT applications. You have more options to choose and code when you connect devices, and ingest, store, and analyze your data. Azure IoT platform services include the products Azure IoT Hub and Azure Digital Twins.
- Azure IoT Central is a managed app platform that lets you quickly evaluate your IoT solution by reducing the number of decisions needed to achieve results. IoT Central takes care of most infrastructure elements in your solution, so you can focus on adding industry knowledge, and evaluating the solution.

IoT Hub tiers

Most IoT solutions require bi-directional communication between devices and the cloud to be fully functional and secure. The basic IoT Hub tier provides core functionality, but excludes bi-directional control. For some early solution implementations, you might be able to reduce costs by using the basic tier. As your solution progresses, you can switch to a standard tier to optimize a secure communication channel for lower cloud-to-device messaging costs. For more information, see Choose the right IoT Hub tier for your solution.

IoT Hub message size and frequency

Messaging costs depend greatly on device *chattiness* and message size. Chatty devices send many messages to the cloud every minute, while relatively quiet devices only send data every hour or more. Avoid device-to-cloud interactions that use many small messages. Clarity about device chattiness and message size helps reduce the likelihood of over-provisioning, which leads to unused cloud capacity, or under-provisioning, which leads to scale challenges. Consider the size and frequency of message payloads to ensure your infrastructure is the correct size and ready to scale.

Avoid cloud-to-device interactions that use many small messages. For example, group multiple device or module twin updates into a single update, which have their own throttling. Be aware of the message size used for the daily quota, 4K-byte for non-free IoT Hub tiers. Sending smaller messages leaves some capacity unused, while larger messages are charged in 4-KB chunks.

Use a single direct method to get direct feedback. Use a single device or module twin status update to exchange configuration and status information asynchronously.

> **Tip**
>
> You can monitor chatty interactions by using **Microsoft Defender for IoT on Azure IoT Hub** and the **Defender for IoT micro agent**. You can create IoT Hub **custom alerts** for device-to-cloud or cloud-to-device interactions that exceed a certain threshold.

If message size is critical to cost management, reducing overhead is especially important with long device lifecycles or large deployments. Options to reduce this overhead include:

- Use a shorter device ID, module ID, twin name, and message topic to reduce the payload in MQTT packets. An MQTT payload looks like `devices/{device_id}/modules/{module_id}/messages/events/`.
- Abbreviate the fixed length overhead and the message.
- Compress the payload, for example by using Gzip.

IoT Hub message quotas and throttling limits

IoT Hub tiers have different sizes with specific quotas and throttling limits for operations. Understand IoT Hub limits and quotas to optimize costs for device-to-cloud and cloud-to-device messaging.

For example, the Standard S1 tier has a daily quota of 400,000 messages. Charges increase in 4-KB chunks based on several factors:

- One device-to-cloud (D2C) message can be up to 4-KB.
- D2C messages that exceed 4-KB are charged in chunks of 4-KB.

- Messages smaller than 4-KB can use the Azure IoT SDK `SendEventBatchAsync` method to optimize batching on the device side. For example, bundling up to four 1-KB messages at the edge increases the daily meter by just one message. Batching is only applicable for AMQP or HTTPS.
- Most operations, such as cloud-to-device messages or device twin operations, also charge messages in 4-KB chunks. All these operations add to the daily throughput and maximum quota of messages.

Review the Azure IoT Hub pricing information documentation for detailed pricing examples.

Besides daily message quotas, service operations have throttling limits. A key part of IoT Hub cost optimization is to optimize both message quotas and operations throttling limits. Study the differences between the limits in the form of operations per second or bytes per second. For more information, see IoT Hub quotas and throttling.

Different throttling limits apply to different IoT Hub operations. Device-to-cloud operations have an operations per second throttle that depends on the tier. In addition to the message size, which is metered in 4-KB chunks, consider the number of operations. Batching on the edge lets you send more messages in a single operation.

A single message of 2-KB, a batched message of 10-KB, or a batched message of 256-KB only counts as a single operation, letting you send more data to the endpoint without reaching throttling limits.

IoT Hub autoscaling

Dynamically adjusting the number of IoT Hub units helps optimize costs when message volume fluctuates. You can implement an autoscale service that automatically monitors and scales your IoT Hub service. See Auto-scale your Azure IoT Hub for a customizable sample to implement autoscale capability. You can use your own custom logic to optimize IoT Hub tier and number of units.

Deployment stamps for scaling

Deployment stamping is a common design pattern for flexible deployment strategies, predictable scale, and cost. This pattern provides several advantages for IoT solutions, such as geo-distributing groups of devices, deploying new features to specific stamps, and observing cost per device. For more information, see Scale IoT solutions with deployment stamps.

Device management and modeling layer

Managing devices is a task that orchestrates complex processes such as supply chain management, device inventory, deployment, installation, operational readiness, device update, bi-directional communication, and provisioning. Device modeling can reduce management costs and messaging traffic volumes.

IoT Plug and Play

For TCO reduction, consider extended use cases as part of platform selection. IoT Plug and Play enables solution builders to integrate devices with IoT Hub or Azure Digital Twins without any manual configuration. IoT Plug and Play uses the Digital Twins Definition Language (DTDL) V2. Both are based on open W3C standards such as JSON-LD and RDF, which enables easier adoption across services and tooling.

There's no extra cost for using IoT Plug and Play and the DTDL. Standard rates for IoT Hub, Azure Digital Twins and other Azure services remain the same.

For more information, see How to convert an existing device to be an IoT Plug and Play device.

IoT Hub DPS

IoT Hub DPS is a helper service for IoT Hub that enables low-cost, zero-touch, just-in-time provisioning to the correct IoT hub without requiring human intervention. DPS enables secure and scalable provisioning of millions of devices to reduce error and cost.

DPS enables low or no-touch device provisioning, so you don't have to train and send people on site. Using DPS reduces the cost for truck rolls and time spent on training and configuration. DPS also reduces the risk of errors due to manual provisioning.

DPS supports device lifecycle management with IoT Hub through enrollment allocation policies, zero-touch provisioning, initial configuration setting, reprovisioning, and de-provisioning. For more information, see:

- DPS pricing
- How to reprovision devices

Asset and device state modeling

Compare cost differences between several device topology and entity stores such as Azure Cosmos DB, Azure Digital Twins, and Azure SQL Database. Each service has a different cost structure and delivers different capabilities to your IoT solution. Depending on required usage, choose the most cost-efficient service.

- Azure Digital Twins can implement a graph-based model of the IoT environment for asset management, device status, and telemetry data. You can use Azure Digital Twins as a tool to model entire environments, with real-time IoT data streaming, and merge business data from non-IoT sources. You can build custom ontologies, or use standards based ontologies such as RealEstateCore, CIM, or NGSI-LD to simplify data exchange with third parties. Azure Digital Twins has a pay-per-use pricing model with no fixed fee.

- Azure Cosmos DB is a globally distributed, multi-model database. Cost is affected by storage and throughput, with regional or globally distributed and replicated data options.

- Azure SQL Database can be an efficient solution for device and asset modeling. SQL Database has several pricing models to help you optimize costs.

Asset deployment model

You can deploy edge solutions with different architectures: multiple endpoints, IoT devices, direct-connected to the cloud, or connected through an edge and/or cloud gateway. Different options for sourcing edge devices can affect TCO and time to market. Ongoing maintenance and support of the device fleet also affects the overall solution cost.

Where data is stored and processed in a given IoT solution affects many factors such as latency, security, and cost. Analyze each use case and examine where it makes most sense to use edge processing and data storage, and how it affects costs. Storing and processing data at the edge can save storage, transportation, and processing costs. But when you take scale into account, cloud services are often better options because of cost and development overhead.

The Azure pricing calculator is a useful tool to compare these options.

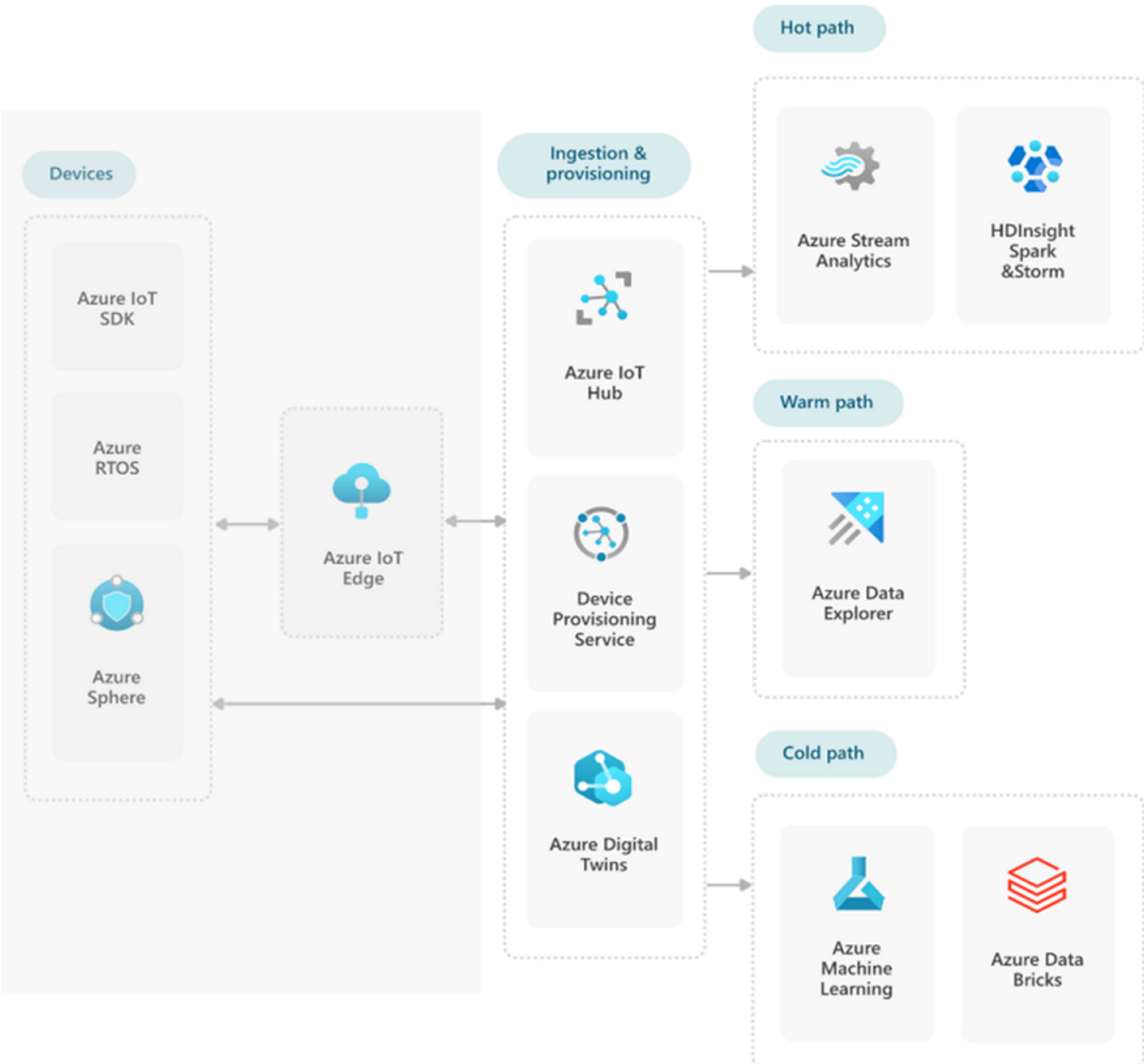
Event processing and analytics layer

The purpose of the event processing and analytics layer is to enable data-driven decisions. Event timing and the purpose of analytics are key factors to consider. The right service choice increases architectural efficiency and reduces the cost of processing data and events.

Based on your requirements, implement hot, warm, or cold path processing for IoT data analytics. The Azure IoT reference architecture helps you understand the difference between these analytics paths and reviews the available analytics services on each path. The Cost Optimization pillar in the Azure Well-Architected Framework includes the cost considerations for data analytics stores, considering data storage, multiple servers to maximize scalability, and the ability to access large data as external tables.

To get started, determine which types of data go through the hot, warm, or cold path:

- Hot path data is kept in memory and analyzed in near real-time, typically using stream processing. The output may trigger an alert or be written to a structured format that analytics tools can query immediately.
- Warm path data, such as from the last day, week, or month, is kept in a storage service that can be queried immediately.
- Cold path historical data is kept in lower-cost storage to be queried in large batches.



Storage layer

One of the goals of an IoT solution is to provide data to end users. It's important to understand storage types, capacity, and pricing to create a strategy for optimizing storage costs.

Storage types

The choice of a repository for telemetry depends on the use case for your IoT data. If the purpose is just to monitor IoT data, and volumes are low, you can use a database. If your scenario includes data analysis, you should save telemetry data to storage. For time series optimized, append-only storage and querying, consider purpose-designed solutions such as Azure Data Explorer.

Storage and databases aren't mutually exclusive. Both services can work together, especially with well-defined hot, warm, and cold analytics paths. Azure Data Explorer and databases are commonly used for hot and warm path scenarios.

For Azure Storage, it's also important to consider data lifecycle factors like access frequency, retention requirements, and backups. Azure Storage helps you define the data lifecycle and automate the process of moving data from the hot tier to other tiers, which reduces long-term storage costs. For more information, see Configure a lifecycle management policy.

Database solutions

For database capabilities, it's common to choose between SQL and no-SQL solutions. SQL databases are best suited for fixed schema telemetry with simple data transformation or data aggregation requirements. To learn more, see Types of databases on Azure .

Azure SQL Database and TimescaleDB for PostgreSQL are common choices for SQL database. For more information, see the following articles:

- Plan and manage costs for Azure SQL Database
- Azure SQL Database and cost optimization
- Azure SQL Database for PostgreSQL Extension
- Performance tuning for Azure SQL Databases

If the data is best represented as an object or document without a fixed schema, no-SQL is a better option. Azure Cosmos DB provides multiple APIs such as SQL or MongoDB.

For any database, partition and index strategies are important for performance optimization and reducing unnecessary costs. For more information, see:

- Partitioning and horizontal scaling in Azure Cosmos DB
- Plan and manage costs for Azure Cosmos DB

Azure Synapse Analytics is a modern Azure data warehouse. Synapse Analytics scales by Data Warehouse Units (DWU), and you should choose the right capacity to handle your solution requirements. Depending on use case, you can pause compute when no job is running to reduce operational costs.

Transport layer

The transport layer transfers and routes data between other layers. As data travels between layers and services, the choice of protocol affects costs. Use cases such as field gateways, industry open protocol, and IoT network selection also affect costs in the transport layer.

To reduce transmission sizes and costs, choose the right protocol for your IoT devices to send telemetry.

Device clients regularly send *keep-alive* messages to IoT Hub. According to Charges per operation, there's no charge for keep-alive messages. But you don't need to add a keep-alive property in the telemetry if there's no specific requirement for it. For flexibility, some Azure IoT Device SDKs provide the option to set a timespan for these messages if you're using the AMQP or MQTT protocols.

For battery-powered IoT devices, you can choose between keeping connections alive or reconnecting when the devices wake up. This choice affects power consumption and network costs.

Reconnecting consumes packets around 6-KB for TLS connection, device authentication, and retrieving a device twin, but saves battery capacity if the device wakes up only once or twice per day. You can bundle messages together to decrease TLS overhead. Keeping alive consumes hundreds of bytes, but keeping the connection alive saves network costs if the device wakes up every few hours or less.

For high-level guidance about the connectivity and reliable messaging features in Azure IoT device SDKs, see Manage connectivity and reliable messaging by using Azure IoT Hub device SDKs. This guidance helps you reduce the costs of handling unexpected behavior between device and Azure IoT services.

DPS reduces device lifecycle management costs from zero-touch provisioning to retirement, but connecting to DPS consumes network cost for TLS and authentication. To reduce network traffic, devices should cache IoT Hub information during provisioning, and then connect to IoT Hub directly until they need to reprovision. For more information, see Send a provisioning request from the device.

Interaction and reporting layer

As IoT handles time-series data, there are many interactions from a large number of devices. Reporting and visualizing realizes the value of this data. Intuitive and simplified user experiences and well-designed data interactions can be costly to build.

Grafana is an open-source data visualization tool that provides optimized dashboards for time-series data. Grafana communities provide examples that you can reuse and customize in your environment. You can implement metrics and dashboard from time-series data with little effort. Azure provides a Grafana plug-in for Azure Monitor.

Reporting and dashboard tools like Power BI allow a quick start from unstructured IoT data. Power BI provides an intuitive user interface and capabilities. You can easily develop dashboards and reports using time-series data, and get the benefits of security and deployment at low cost.

Integration layer

Integration with other systems and services can be complex. Many services can help maximize efficiency to optimize costs in the integration layer.

Azure Digital Twins can integrate various systems and services with IoT data. Azure Digital Twins transforms all data into its own digital entity, so it's important to understand its service limits and tuning points for cost reductions. Review Azure Digital Twins service limits when designing your architecture. Understand functional limitations to help integrate effectively with business systems.

When you use the query API, Azure Digital Twins charges per Query Unit (QU). You can trace the number of QUs the query consumed in the response header. Reduce query complexity and the number of results to optimize costs. For more information, see Find the QU consumption in Azure Digital Twins.

DevOps layer

Cloud platforms transform capital expenditure (CAPEX) to operational expenditure (OPEX). While this model provides flexibility and agility, you still need a well-defined deployment and operational model to take full advantage of the cloud platform. A well-planned deployment creates repeatable assets to shorten time to market.

A cloud platform provides agility for developers to deploy resources in seconds, but there's a risk of provisioning resources unintentionally, or over-provisioning. A proper cloud governance model can minimize such risks and help avoid unwanted costs.

Development environments

Developers can take advantage of the flexibility that Azure provides to optimize development cost. The IoT Hub free tier, limited to one instance per subscription, offers standard capabilities but is limited to 8000 messages a day. This tier is sufficient for early-stage development with a limited number of devices and messages.

For compute environments, you can adopt serverless architecture for cloud-native IoT solutions. Some popular Azure services for IoT workloads include Azure Functions and Azure Stream Analytics. The billing mechanism depends on the service. Some services, like Azure Stream Analytics for real-time processing, let developers pause services without incurring extra costs. Other services bill by usage. For example, Azure Functions bills based on number of transactions. Developers can take advantage of these cloud-native capabilities to optimize both development and operational cost.

An integrated development environment (IDE) accelerates development and deployment. Some open-source IDEs like Visual Studio Code provide Azure IoT extensions that let developers develop and deploy code to Azure IoT services at no cost.

Azure IoT provides free GitHub code samples with guidance. These samples help developers extend device, IoT Edge, IoT Hub, and Azure Digital Twins applications. GitHub also has features to implement seamless continuous integration and continuous deployment (CI/CD) environments with low cost and effort. GitHub Actions are free for open-source projects. For more information, see GitHub plans and features.

Load testing for cost estimation

You can use load testing to estimate overall costs, including cloud services, for end-to-end IoT solutions. Because IoT solutions use large amounts of data, a simulator can help with load testing. Simulation code samples like the Azure IoT Device Telemetry Simulator help you test and estimate costs at scale with various parameters.

Deployment environments

It's common to deploy workloads in multiple environments, such as development and production. Through infrastructure-as-code (IaC), you can accelerate deployment and reduce time to market by reusing code. IaC can help avoid unintentional deployments such as incorrect tiers. Azure services like Azure Resource Manager and Azure Bicep, or third-party services such as Terraform and Pulumi, are common IaC options.

You can apply DevOps deployment practices to IoT solutions by using build and release pipelines to different environments. For an example, see Use a DevOps pipeline to deploy a predictive maintenance solution.

Support and maintenance

Long-term support and maintenance of field devices can escalate to become the largest cost burden for a deployed solution. Careful consideration of system TCO is crucial to realizing Return on Investment (ROI).

You need to support and maintain IoT devices for the lifetime of the solution. Tasks include hardware repairs, software upgrades, OS maintenance, and security patching. Consider ongoing licensing costs for commercial software and proprietary drivers and protocols. If you can't do remote maintenance, you need to budget for onsite repairs and updates. For hardware repairs or replacements, you must keep suitable spares in stock.

For solutions that use cellular or paid connectivity media, select a suitable data plan based on the number of devices, the size and frequency of data transmissions, and device deployment location. If you have a service level agreement (SLA), you need a cost-effective combination of hardware, infrastructure, and trained staff to meet the SLA.

Cloud governance

Cloud governance is essential for compliance, security, and preventing unnecessary costs.

- Cost management APIs let you explore cost and usage data through multidimensional analysis. You can create customized filters and expressions that help answer Azure resource consumption-related questions. Cost management APIs can generate alerts when consumption reaches configured thresholds. Cost management APIs are available for IoT Central, IoT Hub, and DPS.

- Resource tagging applies labels to deployed resources. Along with Azure Cost Management, tagging provides insights on ongoing costs based on the labels. For more information, see Common cost analysis uses.

- Azure Policy comes with built-in policies to label resources automatically, or flag resources without tagging. To learn more, see Assign policy definitions for tag compliance. Another use case for Azure Policy is to prevent provisioning of certain tiers, which helps prevent over-provisioning in development or production environments.

Monitoring

Many tools included in your Azure subscription can help your organization implement financial governance and get more value out of your IoT services. These tools help you track resource usage and manage costs across all of your clouds with a single, unified view. You can access rich operational and financial insights to make informed decisions.

Telemetry logging commonly uses Log Analytics workspaces in Azure Monitor. Log Analytics includes 5 GB of storage, and the first 30 days of retention are free. Depending on business needs, you might need a longer retention period. Review and decide the right retention period to avoid unintentional costs.

Log Analytics provides a workspace environment to query logs interactively. You can export logs periodically to external locations such as Azure Data Explorer, or archive logs in a storage account for a less expensive storage option. For more information, see Monitor usage and estimated costs in Azure Monitor.

Azure Advisor

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Advisor analyzes your resource configuration and usage telemetry, and recommends solutions that can help you improve cost effectiveness, performance, reliability, and security.

Advisor helps you optimize and reduce your overall Azure spending by identifying idle and underutilized resources. You can get cost recommendations from the cost tab on the Advisor dashboard.

Although Advisor doesn't offer specific recommendations for IoT services, it can provide useful recommendations for Azure infrastructure, storage, and analytics services. For more information, see Reduce service costs by using Azure Advisor.

Next steps

Operational excellence in your IoT workload

Related resources

- Cost optimization design principles
- Capture cost requirements
- Develop a cost model
- Checklist - Design for cost
- Checklist - Optimize cost
- Azure IoT reference architecture
- Azure IoT documentation

Operational excellence in your IoT workload

Article • 04/27/2023

Given the complexity of IoT solutions requirements, organization's operational capabilities are important for driving sustainable business value. This guide focuses on the operational aspects of IoT devices and services that uniquely address the core requirements of an IoT solution.

Operational excellence in an IoT workload requires full visibility and control over all hardware and software components of the solution. Design, development, provisioning, monitoring, support, and maintenance practices must be agile and deliver business value without increasing operational risk.

In IoT solutions, the device diversity and scale, different network types, and geographically distributed locations significantly shift the cloud and hybrid shared responsibility model away from the cloud provider. Cloud services make it easier for organizations to operate IoT devices and networks themselves or by using third parties, but the organizations themselves own the operational responsibility for these key elements of IoT workloads.

Operational excellence ensures that your IoT solution can successfully:

- Support different user roles.
- Manage all device lifecycle stages.
- Scale efficiently to meet changes on demand.
- Use automation for management and monitoring.
- Integrate with other back-end systems.

Assess operational excellence in your IoT workload

To assess your IoT workload through the lenses of the Well-Architected Framework Operational Excellence pillar, complete the operational excellence questions for IoT workloads in the Azure Well-Architected Review. After the assessment identifies key operational excellence recommendations for your IoT solution, use the following content to help implement the recommendations.

Design Principles

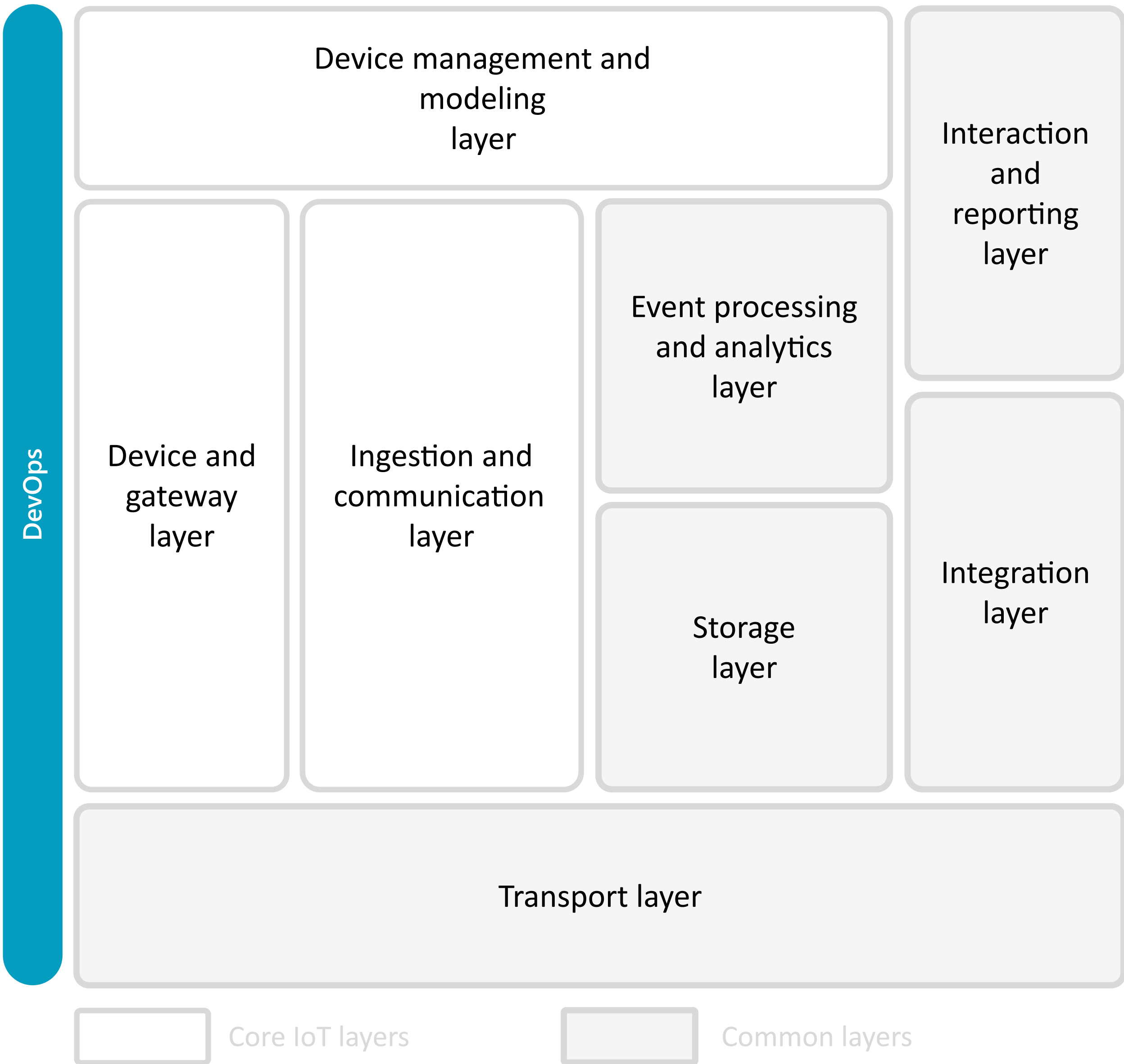
Five pillars of architectural excellence underpin the IoT workload design methodology. These pillars serve as a compass for subsequent design decisions across the key IoT design areas. The following design principles extend the quality pillar of the Azure Well-Architected Framework - Operational Excellence.

Design principle	Considerations
Embrace continuous operations and scaling	Ensure that the IoT solution can successfully manage automated device provisioning, integrate with other backend systems, support different roles such as solution developers, solution administrators, and operators, and adapt and scale efficiently to any changes on demand such as new IoT devices being deployed or higher ingestion throughput.
Optimize build and release processes	Any successful enterprise IoT solution requires a strategy to establish and update a device or fleet of device's configuration. A device's configuration includes device properties, connection settings, relationships, and firmware. IoT operators require simple and reliable tools that enable them to update a device or fleet of device's configuration at any point during the device's lifetime.
Understand operational health	Use IoT solution logging, monitoring, and alerting systems to determine whether the solution is functioning as expected and to help troubleshoot problems throughout the lifecycle of the solution.
Use automation and DevOps	An IoT device is fundamentally a small computer with specialized hardware and software. IoT devices are often constrained in hardware, for example having limited memory or compute capacity. Automation and DevOps are essential to ensure that OS and software for IoT devices and gateways are properly uploaded and deployed to minimize operational downtime. Automation and DevOps are essential for monitoring and managing the lifecycle of IoT devices.

IoT architecture layers

Operational Excellence design principles help clarify considerations to ensure your IoT workload meets requirements across the foundational IoT architecture layers.

The IoT core layers: Device and gateway, device management and modeling, and ingestion and communication, identify IoT-specific solutions. The other layers and cross-cutting activities are also common to, and often shared with, other workloads. DevOps cross-cutting activities are especially important to support the operational excellence pillar.



Device and gateway layer

This layer represents the physical or virtual device and gateway hardware deployed at the edge or on premises.

A key factor in IoT operational excellence is an organization's ability to plan, provision, configure, monitor, and retire IoT devices. Organizations must select IoT hardware that meets business and technical requirements, and define appropriate testing procedures to ensure operational reliability.

Greenfield projects that use new hardware usually have more flexibility in device types, firmware and connectivity features, and technical specifications. You might need to select devices that comply with regional certification requirements or regulations such as CE, FCC, UL, PCI, or FDA.

Brownfield projects that already have hardware deployed typically have more hardware restrictions. You might need to look for other types of hardware, such as protocol or identity translation devices, or connectivity gateways such as Bluetooth to MQ Telemetry Transport (MQTT) gateway.

Azure Certified Device Program certification validates that a device can connect with Azure IoT Hub and securely provision through the IoT Hub Device Provisioning Service (DPS). The Azure Certified Device Catalog can help you find and select certified partner hardware. The Device Catalog has search and filter capabilities you can use to find hardware that meets your solution requirements.

An important feature to look for in Azure IoT-certified hardware is Azure Plug-and-Play and Digital Twins Definition Language (DTDL) compatibility. These features ensure that devices integrate seamlessly with services such as Azure Digital Twins. For Azure IoT Edge scenarios, it's important to find catalog devices that have the IoT Edge Managed certification. This certification guarantees the device can run the IoT Edge runtime, and enables deployment and management of IoT Edge modules that support edge processing and analytics workloads.

Device components and spares must be available to cover maintenance and support contracts for the lifetime of the solution. Ensure a timely and secure equipment supply at the start of the project, because this requirement can be expensive to introduce later. Use a trusted vendor chain and consider dual or multiple supply sources.

Ingestion and communication layer

The organization's network operations team typically partners with the telecommunication operator to handle the communication network technology stack of an IoT workload. Coordinate with your telecommunication operator to set up and operate the wired and wireless communication network components of your IoT solutions and operations.

Capacity scaling

Configure the ingestion and other back-end layers of the IoT cloud solution to be able to scale to handle expected and unexpected capacity needs. If your solution is tied to a connected product, you must handle fluctuations in expected load. Load can be impacted by marketing initiatives such as sales or promotions, or by seasonal events such as holidays. You should test load variations prior to events, including unexpected events, to ensure that your IoT solution can scale.

Azure offers several options to meet capacity requirements as your business grows. Capacity planning and scaling for your IoT solution varies depending on whether you build an IoT Central or IoT Hub -based solution.

- IoT Central is a managed application platform that you can use to quickly evaluate your IoT scenario and assess the opportunities for your business. IoT Central takes care of most infrastructure elements however, it stores only 30 days of data. Because most IoT solutions export data to other services, you should focus on making sure those other services can handle expected and unexpected capacity needs during the evaluation of your solution.

- With an IoT Hub-based solution, it's your responsibility to scale up to handle growth in the number of messages being ingested and to scale out to handle regional demands. Understanding the number of messages that devices will send to IoT Hub and the sustained throughput is critical to selecting the correct IoT Hub tier to support the predicted demand.

  If you're approaching the IoT Hub message limit, your system should be able to automatically scale up IoT Hub to the next unit of capacity. Any back-end services in the IoT solution, such as Azure Stream Analytics, Azure Cosmos DB, and Azure Data Explorer must support scalability to ensure there are no bottlenecks anywhere in the solution's data flow.

You should also plan for edge device capacity needs and requirements. Whether you're managing real-time operating system (RTOS)-based devices or larger compute devices with IoT Edge, make sure compute and memory sizing are adequate for your specific use cases.

Device management and modeling layer

Implement a centralized device management solution to administer, monitor, and operate the lifecycle of IoT devices, and to manage the overall configuration of the IoT solution. Consider implementing an integrated UI to assist operation teams with device fleet management.

Device provisioning

Define a remote device provisioning strategy to enable zero-touch, just-in-time provisioning of IoT devices in the field without requiring human intervention.

For remote provisioning of IoT devices, Azure IoT Hub Device Provisioning Service (DPS) enables connecting and configuring remote devices to IoT Hub. DPS enables zero-touch provisioning without hard-coding information at the factory, and enables load-balancing of devices across multiple IoT hubs.

Although DPS supports symmetric key attestation, in a production environment you should use either the X.509 certificate or TPM attestation mechanisms. If you use X.509 certificates, you should deploy the root certificate, or an intermediate certificate signed by the root certificate to DPS, to allow devices in the field properly authenticate to the service and be assigned to their correct IoT hub.

Part of an IoT solution lifecycle includes reprovisioning devices in the field or moving them between IoT hubs. DPS enables the configuration of reprovisioning policies that determine expected behavior when an IoT device submits a new provisioning request. Devices should be programmed to send a provisioning request on reboot, and should implement a method to manually trigger provisioning on demand. This mechanism ensures that every time a device starts up, it contacts DPS to get redirected to the appropriate IoT hub.

Device configuration and update management

Establish a strategy to update device or device fleet configuration. A device's configuration includes device properties, firmware, connection settings, and relationships. IoT operators need simple and reliable tools that let them update a device or device fleet's configuration at any point during the device's lifetime.

An IoT solution's scale and specific use of a device's configuration, influences the design of a configuration management strategy. It's important to automate this strategy as much as possible, and to ensure that the configuration can be set and updated efficiently.

A configuration management strategy should support:

- Inventory of IoT devices and IoT Edge devices deployed in the field.
- Gradual update rollout through device grouping.
- Resilient updates to support testing and rollbacks.
- Automatic updates for existing or new devices.
- Updated status reports and alerts.

Azure features that support these configuration management requirements include IoT Hub automatic device management, IoT Edge automatic deployments, IoT Hub scheduled jobs, and Device Update for IoT Hub.

- For continuous updates to existing or new devices and IoT Edge device configurations, such as properties, application specific settings, or relationships, use either IoT Hub automatic device management or IoT Edge automatic deployments. Both features offer an efficient, secure, and reliable way to automate configuration deployments for a fleet or specific group of devices. The services continuously monitor all new and existing targeted devices and their configuration based on tags, to ensure the devices always have the specified configuration. The key difference between these features is that automatic device management applies only to non-IoT Edge devices, and IoT Edge automatic deployments apply only to IoT Edge devices.

- To update an existing device or IoT Edge device configuration based on a one-time or recurring schedule, use IoT Hub scheduled jobs. This feature is an efficient, secure, and reliable way to provide a configuration update for a fleet or specific group of devices at a scheduled time.
- To update existing device or IoT Edge device firmware, application, or package updates over-the-air (OTA), use Device Update for IoT Hub. This service is a safe, secure, and reliable way to update a fleet or specific group of devices.

It's a good idea to have a manual update method for IoT devices. Due to root certificate changes or connectivity issues, you may need to manually update devices by physically connecting to a local computer or using a local connectivity protocol such as Bluetooth.

To learn more about device management, see:

- Azure IoT Hub Automatic Device Management
- Azure IoT Edge automatic deployments
- Azure IoT Hub scheduled jobs
- Device Update for IoT Hub

Management user interface

Solution operators and administrators need an interface to interact with the IoT solution, for example provision devices, add or remove users, send commands to IoT devices, or manage device updates.

IoT Central has a built-in, easy-to-use management interface that lets operators and administrators focus on adding industry knowledge, and evaluating the solution.

When you build your enterprise solution by using the platform services, such as IoT Hub and Azure Digital Twins you can build a custom management UI by using the REST APIs exposed in IoT Hub REST APIs and Azure Digital Twins REST APIs.

Integration layer

A typical IoT solution is composed of multiple components such as ingestion, routing, data storage, and data processing. It's important to document and have a good understanding of the entire data flow of the IoT solution. Have testing procedures in place to ensure the different parts of the solution work as expected and meet the technical and operational requirements of the organization. Implement automation to identify device capabilities at scale as they connect to your IoT solution and to easily integrate with back-end services.

Configure and test reliable integration with other Azure and third-party services that support the back-end and front-end services of the IoT application. A successful IoT implementation requires integrating IoT services such as IoT Hub and DPS with other Azure and third-party services.

For example, DPS supports custom allocation policies by using custom code and Azure Functions, so it's important to confirm that the Azure Function allows traffic coming from DPS and IoT Hub. Another example is the integration between IoT Hub and backend services to enable features such as message routing and file upload. IoT Hub needs to properly authenticate to those Azure services. You should use managed identities to eliminate the need to manage those credentials manually.

DevOps layer

DevOps includes role and user management, metrics collection, monitoring, and automation.

Role and user management

A key decision early in a solution design phase is to define the roles that implement and manage the solution. Determine the roles that are responsible for developing, managing, and operating the IoT solution at scale, and the users assigned to those roles.

Ideally, the solution should trust a centralized identity provider, such as Azure Active Directory (Azure AD), and only let the appropriate users in those roles perform management or operation activities, such as creating and provisioning new devices, sending commands to hardware in the field, deploying updates, and modifying user permissions.

In an IoT Hub-based solution, you can use Azure AD to authenticate requests to IoT Hub service APIs, such as creating device identities or invoking direct methods. You can develop a custom management UI for solution operators and administrators, that authenticates users against Azure AD and executes API requests to the IoT solution back end on behalf of those users.

IoT Edge Metrics Collector

Azure IoT Edge provides the IoT Edge Metrics Collector ready-to-use IoT Edge module in the IoT Edge Module Marketplace. Add this module to an IoT Edge deployment to collect metrics and send them to Azure Monitor. The open-source module code is a multi-architecture Docker container image that supports Linux x64, ARM32, and ARM64 version 1809.

The Metrics Collector module can collect logs from all the modules that can emit metrics by using the Prometheus data model . While built-in metrics enable broad workload visibility by default, you can also use custom modules to emit scenario-specific metrics that enhance the monitoring solution.

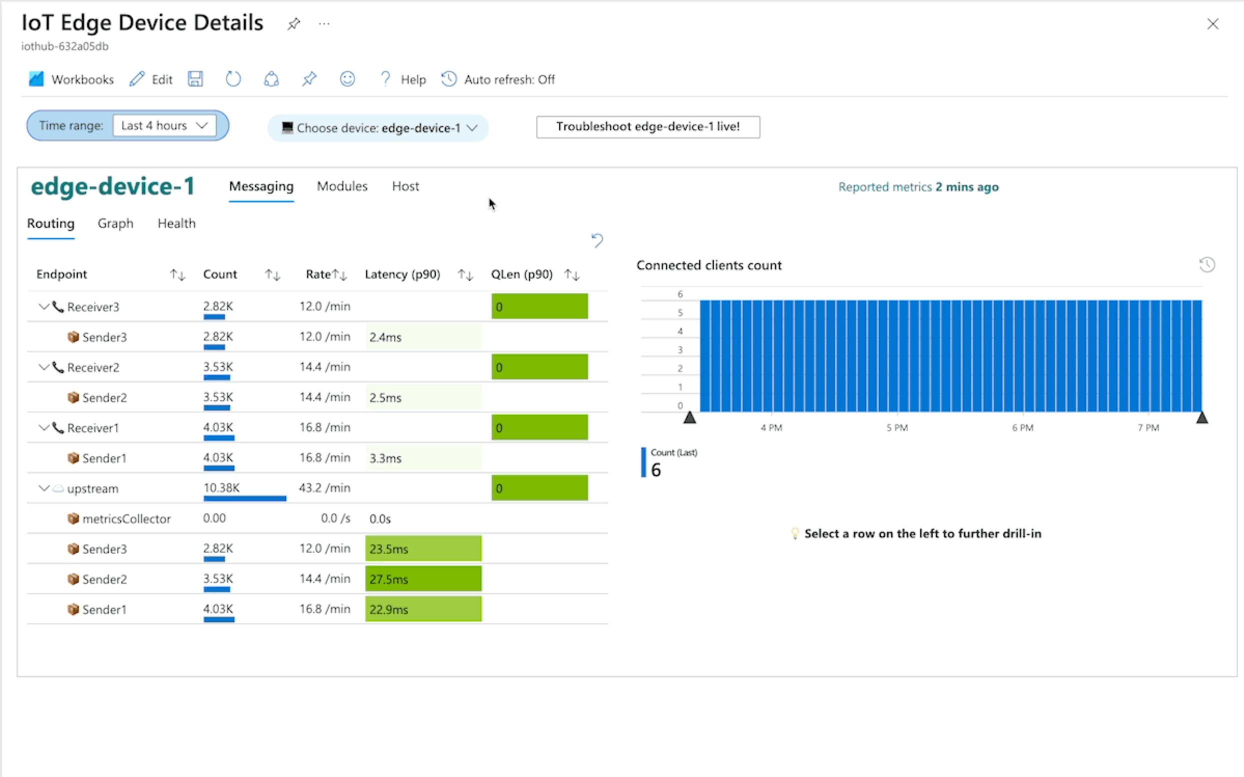
There are two options to send metrics from the Metrics Collector module to the cloud:

- Send the metrics to Log Analytics. The collected metrics are ingested into the specified Log Analytics workspace using a fixed, native table called `InsightsMetrics`.
- Send the metrics to IoT Hub. You can configure the collector module to send the collected metrics as UTF-8 encoded JSON device-to-cloud messages through the edge hub module. This option unlocks monitoring of locked-down IoT Edge devices that are only allowed external access to the IoT Hub endpoint.

The `AllowedMetrics` and `BlockedMetrics` configuration options take space- or comma-separated lists of metric selectors. A metric is matched to the list and included or excluded if it matches one or more metrics in either list.

You can visually explore metrics collected from IoT Edge devices by using Azure Monitor workbooks. Curated workbooks use built-in metrics from the IoT Edge runtime that are ingested into a Log Analytics workspace. These views don't need any metrics instrumentation from the workload modules.

The Azure portal provides curated monitoring workbooks for IoT Edge devices as public templates. To access the workbooks, from your **IoT Hub** or **IoT Central** page in the Azure portal, navigate to the **Workbooks** page in the **Monitoring** section.



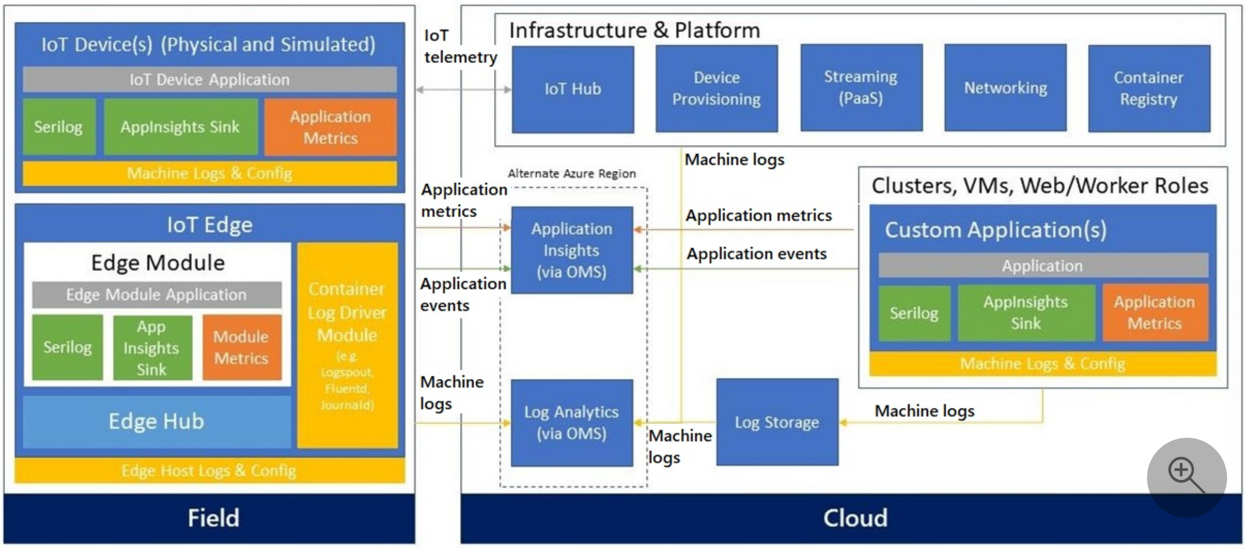
Monitoring

Use IoT solution logging, monitoring, and alerting systems to determine whether the solution is functioning as expected and to help troubleshoot and mitigate problems. Monitoring and logging help determine whether devices or systems are in an error condition, correctly configured, generating accurate data, and meeting defined service level objectives.

IoT logging and monitoring systems can be more complicated than in standard line-of-business applications. The complexity arises because IoT solutions often span:

- Physical sensors that interact with an environment.
- Applications on the edge doing activities like data shaping and protocol translation.
- Infrastructure components such as on-premises gateways, firewalls, and switches.
- Ingestion and messaging services.
- Persistence mechanisms.
- Insight and reporting applications.
- Subsystems that operate and scale independently in the cloud.

The following simplified logging and monitoring architecture shows examples of typical IoT solution components and how they use recommended technologies.



If your critical applications and business processes rely on Azure resources, you should monitor those resources for availability and performance. You can use Azure Monitor to carry out the following monitoring activities:

- Detect and diagnose issues across applications and dependencies with Application Insights.
- Correlate infrastructure issues with VM Insights and Container Insights.
- Drill into your monitoring data with Log Analytics for troubleshooting and deep diagnostics.
- Support operations at scale with smart alerts and automated actions.
- Create visualizations with Azure dashboards and workbooks.
- Collect data from monitored resources using Azure Monitor Metrics.

Monitor IoT Hub

Azure IoT Hub collects the same types of monitoring data as other Azure resources, as described in Monitoring data from Azure resources. The **Overview** page in the Azure portal for each IoT hub includes charts that provide some usage metrics, such as the number of messages used and the number of devices connected to the hub. The information on the **Overview** page is useful, but represents only a small amount of the monitoring data available for an IoT hub.

Some monitoring data is collected automatically and is available for analysis as soon as you create your IoT hub. You can configure other types of data collection. To learn more about the metrics and logs that IoT Hub creates, see Monitoring Azure IoT Hub data reference.

Monitor updates

As with any deployment or update, you should monitor the update state of deployments and devices. DevOps provides a way to consistently deliver fresh software updates. Device Update for IoT Hub monitors compliance by measuring how many devices have installed the highest version compatible update. A device is compliant if it has installed the highest version available compatible update.

Monitor configuration

As with any deployment or update, you should monitor and alert on the status of a device configuration or update deployment. Each Azure IoT configuration service collects and stores logs and metrics in Azure Monitor. You can use this data to create Azure Monitor alerts to send notifications when a configuration deployment or update is created, completed, or failed.

If the monitoring data provided by each of the Azure IoT configuration services isn't enough, the Azure IoT Hub service APIs offer a more granular view.

Monitor automation and DevOps

DPS, IoT Hub, and IoT Edge provide continuous metrics and status updates that are key inputs to monitor continuous integration/continuous deployment (CI/CD) status or automation script output. You can collect and analyze these metrics in a Log Analytics workspace and then define alerts.

To learn more about monitoring, see:

- Monitor device connectivity using the Azure CLI IoT extension
- Monitor, diagnose, and troubleshoot Azure IoT Hub device connectivity
- Monitor Azure IoT Hub
- Check Azure IoT Hub service and resource health
- Tutorial: Set up and use metrics and logs with an Azure IoT hub
- Collect and transport metrics

Automation

An IoT device is fundamentally a small computer with specialized hardware and software. IoT devices are often hardware-constrained, for example have limited memory or compute capacity. Automation and DevOps ensure that IoT device and gateway software is properly uploaded and deployed to minimize operational downtime. Automation and DevOps are essential to monitoring and managing the full lifecycle of developing, deploying, and operating an IoT solution and devices.

The key benefit of a mature DevOps implementation is agility, the ability to quickly sense and respond to changes in business needs. To use automation with DevOps for agile software development, deployment, testing, integration, and operations, follow these recommendations:

- Use CI/CD DevOps principles and processes to boost productivity and create a seamless rapid development cycle.
- Deploy application software changes in an infrastructure-as-code (IaC) environment to automate and manage the ongoing operation of deployed software.
- Automate the IoT application software lifecycle from development through testing to deployment to IT operations.
- Use DevOps tools and processes in IoT Hub and IoT Edge to automate the edge software lifecycle. Use IoT Edge to deploy IoT application software on devices.
- Provide operators with tools to gain visibility and insights, collaborate, control, and maintain a reliable IoT solution.
- Embrace cross-functional teams to deliver continuously for solutions. Device vendors and cross-functional solution developers should work together to develop and deploy IoT solutions.
- Evolve business and deployment models to create possibilities for different business models and pilot validation, deployment, and enhancements.

Automate device lifecycle

Connected IoT Edge devices have a lifecycle that extends beyond deploy, break and fix, and retire. Connected devices put organizations in the best position to capitalize on opportunity and continuously add incremental innovation throughout the system lifecycle.

In IoT solutions, software programs installed on hardware define system functionality. Thousands of devices might be connected to a single cloud endpoint such as IoT Hub. Any change in configuration or software must be spread across all the devices. To change system functionality, update software instead of making hardware changes or local interventions.

When you implement automation and DevOps in IoT systems, follow specific automation and DevOps requirements for each device lifecycle phase. The following tables describe Azure IoT features that support three phases of the device lifecycle.

Beginning of life

Expectations	Platform feature available with code snippets
Non-DPS device registration	Bulk device updates
Device provisioning	DPS configuration required to provide zero touch device provisioning
Device certificate and token management	Control access to IoT Hub using Shared Access Signatures (SAS)
Device certificate lifecycle management	CA certificate lifecycle management with DPS and DigiCert
Device initial configurations	Device twins and device modules

Midlife

Expectations	Platform feature available with code snippets
Continuous device configuration management at scale	Device twins and device modules
CI/CD pipeline for IoT Edge modules	Continuous integration and continuous deployment (CI/CD) to Azure IoT Edge devices
Device reprovisioning	DPS device reprovisioning
SAS key generation for changes or expiration	Control access to IoT Hub using Shared Access Signatures (SAS)
Log and device diagnostics	Pre-configured Azure workbooks for IoT Hub
Azure IoT Edge monitoring diagnostics	Collect and transport IoT Edge device logs and metrics
OTA device updates	Device Update for IoT Hub

End of life

Expectations	Platform feature available with code snippets
Unenroll devices	Disenroll a device from DPS
Remove device-specific configuration	Device twins and device modules
Device replacement	Same as beginning of life

Next steps

Performance efficiency in your IoT workload

Related resources

- Automatic IoT device and module management using the Azure CLI
- Azure IoT Edge requirements
- Azure IoT Edge managed certification requirements
- DPS X.509 attestation
- Azure IoT Hub scaling
- Control access to Azure IoT Hub by using Azure Active Directory
- Plan for capacity - Microsoft Azure Well-Architected Framework
- Operational excellence design principles
- Azure IoT reference architecture
- Azure IoT documentation

Performance efficiency in your IoT workload

Article • 04/27/2023

IoT solutions include device, edge and cloud components, and range from millions of small devices connected to the cloud to industrial solutions where a few powerful servers are gateways for cloud connectivity. The number of devices, their physical and geographical placement, and the number of messages they send or receive are some of the factors that can define the performance efficiency of an IoT workload.

Performance efficiency also includes an IoT workload's ability to scale efficiently to meet demands. A benefit of the cloud is geographical availability and the ability to scale services on demand, with little or no application downtime.

Performance efficiency represents performance relative to resource use under stated conditions. Performance efficiency measures how well a product or system, when performing its functions, meets requirements for:

- **Time behavior**, such as response times, processing times, and throughput rates.
- **Resource utilization**, or amounts and types of resources used.
- **Capacity**, or maximum limits.

Assess performance efficiency in your IoT workload

To assess your IoT workload through the lenses of the Well-Architected Framework Performance Efficiency pillar, complete the performance efficiency questions for IoT workloads in the Azure Well-Architected Review. After the assessment identifies key performance efficiency recommendations for your IoT solution, use the following content to help implement the recommendations.

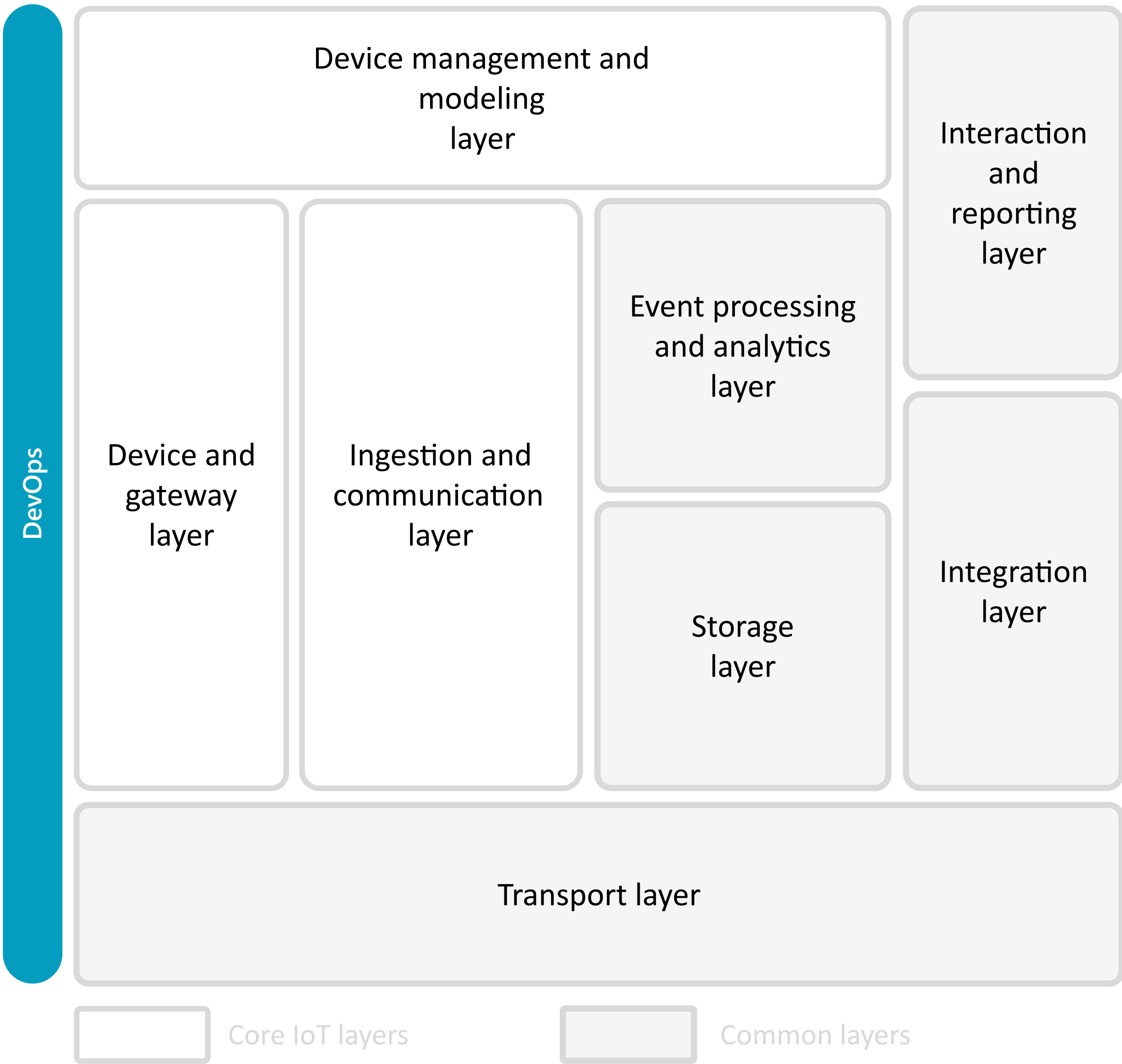
Design principles

Five pillars of architectural excellence underpin the IoT workload design methodology. These pillars serve as a compass for subsequent design decisions across the key IoT design areas. The following design principles extend the quality pillar of the Azure Well-Architected Framework - Performance Efficiency.

Design principle	Considerations
Design for horizontal scaling	An IoT solution can start with a few hundred devices or messages and grow to millions of devices and messages per minute. You can easily scale cloud services to an increase in load, but the situation can be more complex for IoT devices and gateways. IoT devices can be designed or deployed before the solution is finalized. Industrial IoT or similar industries can measure device lifespan in decades. Updating capacity by replacing devices is costly. In these scenarios, it's especially important to plan ahead.
Shift-left on performance testing	Test early and test often to catch issues early. Be aware of the complexity of having sensors, devices, and gateways in geographically different locations with different characteristics, speed, and reliability of communication. Plan for this complexity in your testing, and make sure to test for failure scenarios like network disconnection. Do stress and load testing of all device, edge, and cloud components in your IoT solution.
Continuously monitor for performance in production	To monitor different types of devices in multiple geographical regions, use a distributed monitoring solution. Balance the amount of information monitored and sent to the cloud against memory and performance costs. Tune transmission for diagnostic scenarios, and monitor at multiple levels and layers. Expose gateway metrics for industrial or gateway-enabled solutions.

IoT architecture layers

Performance efficiency design principles help clarify considerations to ensure your IoT workload meets requirements across the foundational IoT architecture layers. The following sections address the layer specifics for the performance efficiency pillar.



Device and gateway layer

An IoT device is a computing device that connects to an IoT solution and can collect, transmit, or receive data. Gateways are connection points between devices and the cloud, or between IoT and other components.

Optimize hardware capabilities

Upgrading or replacing hardware is costly and time consuming. Size IoT devices for required capacity and functionality in advance.

To optimize for hardware capabilities:

- Run compute and input-output intensive tasks on specific hardware. For example, run machine learning (ML) algorithms on local graphics processing units (GPUs).

- Optimize existing hardware capabilities by using efficient languages and frameworks like Embedded C and Rust Embedded. You can use the Azure IoT Embedded C SDK when developing for constrained devices, or when most of the security and communication stack is already available on the device.

- Use the Azure IoT device SDK for C for all you need to connect to the cloud gateway. The Azure IoT Device software development kits (SDKs) manage required message translation, error handling, and retry mechanisms needed for a resilient connection.

Scaling is important for the device and gateway layer. To scale this layer:

- Use gateways as units of scale. If your solution adds IoT devices or assets (for example OPC UA servers) over time, use more edge gateways to ingest data from those servers.

- Conduct a scale assessment for all upstream layers, including cloud gateways and cloud services. To learn more about using multiple IoT hubs as scale units for an IoT solution, see How to provision devices across IoT hubs.

Run workloads at the edge

Depending on system constraints such as network throughput or latency, consider running some workloads at the edge. Separate workloads by time constraint and required latency and response times. Use local compute for low latency and intermittently connected scenarios. Run large-scale workloads in the cloud.

At the edge, use priority queues to send different data streams in the required order. With priority queues messages are sent in order of priority, but Azure IoT Hub still journals messages based on receipt order.

Optimize device connectivity

Consider the following points to optimize device connectivity:

- Use the IoT Hubs that have the lowest latency to your devices. You might need IoT Hubs in multiple regions when devices need to connect from different geographical locations.

- Use an open stateful connection for bi-directional communications between the devices and the IoT solution to minimize the overhead of setting up connections.

- Don't connect all devices at once, for example after a regional power outage. Use truncated exponential backoff with introduced jitter when retrying.

Optimize offline scenarios

You can provide devices with enough information and context to work without a cloud connection and to store data locally, so they can recover from disconnections and reboots. The following strategies support offline operations:

- Ensure the device is capable of storing data locally when the device isn't connected, including logs and cached telemetry according to priority.
- Set a time to live (TTL) on the data, so that expired data is removed automatically.
- Discard less important data when the device isn't connected, to reduce local storage needed and reduce synchronization time when the device reconnects.
- If edge device storage reaches capacity, use a cache eviction strategy such as first-in last-out (FIFO), last-in first-out (LIFO), or priority-based.
- Consider using a separate disk or disk controller to store data, so the device runtime or application can continue to work when low on storage.

Use device twins and module twins to asynchronously sync state information between devices and the cloud, even when devices aren't currently connected to the cloud gateway. Device and module twins contain only the current state at a point in time, not any history or removed information.

Ingestion and communication layer

The data ingestion and communication layer sends data from the devices to the IoT solution. Patterns of communication between devices and the IoT solution include:

- Device-to-cloud messages.
- Cloud-to-device messages.
- File uploads.
- Device twins.
- Direct methods.

Optimize messaging efficiency

The number and size of device to cloud messages is an important parameter for IoT solution performance efficiency. Azure IoT services such as IoT Hub and Azure IoT

Central define message limits per tier, which affects both solution performance and cost.

Consider the following messaging recommendations:

- IoT Hub and IoT Central calculate daily quota message counts based on a 4-KB message size. Sending smaller messages leaves some capacity unused. In general, use message sizes close to the 4-KB boundary. Group smaller device-to-cloud messages into larger messages to reduce the total number of messages, but consider the introduced latency when combining messages.

- Avoid chatty communication. For device-to-device or module-to-module edge communication, don't design interactions that send many small messages.

- Use built-in Azure IoT Edge SDK message batching for Advanced Message Queuing Protocol (AMQP) to send multiple telemetry messages to the cloud.

- Use application-level batching by combining multiple smaller messages at the downstream device and sending larger messages to the edge gateway. This batching limits the message overhead and reduces writes to local edge disk storage.

- Use AMQP connection multiplexing to reduce the dependency on Transmission Control Protocol (TCP) connections limits per SDK client. With AMQP connection multiplexing, multiple devices can use a single TCP connection to IoT Hub.

- Use direct methods for request-reply interactions that can succeed or fail immediately, after a user-specified timeout. This approach is useful for scenarios where the course of action is different depending on whether the device responded.

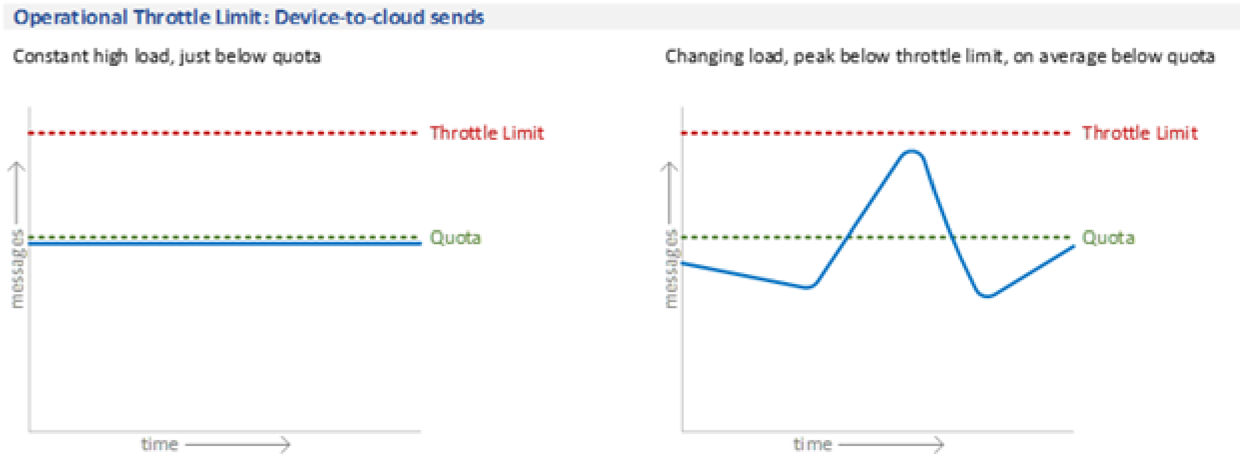
- Use device twins for device state information, including metadata and configurations. IoT Hub maintains a device twin for each device that you connect.

Understand messaging quotas and throttling

The IoT Hub tier sets cloud gateway per-unit limits. The messaging *quota* defines sustained throughput and sustained send rates for the tier. IoT Hub can handle loads above these quotas for short durations to resiliently handle bursts or load overshoots.

Another important limit is the hourly or daily service load or *throttle* limit. Throttle limits protect an IoT hub from too much load for too long a time.

The following diagrams show the relationship between load, quota, and throttle limits. The left diagram shows that IoT Hub can handle sustained or constant high load up to the level of the quota for the IoT Hub tier. The right diagram shows that IoT Hub can handle load that is changing over time, as long as it doesn't hit the throttle limit and on average isn't above the quota for the IoT Hub tier.



Optimize message processing

Messages from a device or gateway might need to be translated, processed, or enriched with more information before storage. This step could be time-consuming, so it's important to evaluate the effect on performance. Some recommendations conflict, such as using compression for optimizing data transfer versus avoiding cloud processing in decrypting messages. These recommendations need to be balanced and evaluated against other architectural pillars and solution requirements.

To optimize cloud data processing performance:

- Optimize the data format used to send data to the cloud. Compare performance (and cost) of bandwidth vs. performance improvement with less cloud data processing needed. Consider using IoT Hub message enrichment to add context to device messages.

- Do time-critical event processing on ingested data as it arrives, instead of storing unprocessed data and requiring complex queries to acquire the data. For time-critical event processing, consider the impacts of late arrival and windowing. Evaluate depending on use case, for example critical alarm handling versus message enrichment.

- Select the right IoT Hub tier, Basic or Standard, based on solution requirements. Be aware of features that the Basic tier doesn't support.

- Select the right IoT Hub tier size, 1, 2, or 3, and the number of instances based on data throughput, quotas, and operation throttles. For IoT Central, select the right tier: Standard 0, Standard 1 or Standard 2, based on the number of messages sent from devices to the cloud.

- Consider using Azure Event Grid for publish-subscribe event routing. For more information, see React to IoT Hub events by using Event Grid to trigger actions and Compare message routing and Event Grid for IoT Hub.

Prioritize data

Some data that devices send to the cloud might be more important than other data. Classifying and handling the data based on priority is a good practice for performance efficiency.

For example, a thermostat sensor sends temperature, humidity, and other telemetry, but also sends an alarm when temperature is outside a defined range. The system classifies the alarm message as higher priority and handles it differently than the temperature telemetry.

Consider the following recommendations for data classification and handling:

- Use IoT Edge priority queues to make sure important data is prioritized while sending to IoT Hub. IoT Edge buffers messages when there's no connectivity, but after the connection is restored, sends all buffered messages in priority order first, followed by new messages.

- Use IoT Hub message routing to separate routes for different data priorities depending on use case. IoT Hub message routing adds some latency.

- Save and send low priority data at longer intervals, or by using batch or file uploads. Malware detection on uploaded files increases latency.

- Separate messages based on time constraints. For example, send messages to IoT Hub directly when there's a time constraint, and utilize file upload via IoT Hub or batch data transfer like Azure Data Factory if there's no time constraint. You can use the IoT Edge blob module for file upload.

Device management and modeling layer

Different types of devices can connect to an IoT solution, and an IoT solution can connect to many devices and gateways at the same time. Besides connecting and configuring devices and gateways, the IoT solution must understand the data the devices and gateways capture and ingest, and must transfer and contextualize that data.

IoT components can use different protocols, connectivity, data ingestion frequencies, and communication patterns. The IoT solution must be able to manage which devices and gateways are connected and how they're configured.

To manage devices and configurations for performance efficiency:

- Optimize sizing based on device and message load.
- Know the number of messages the cloud gateway can handle, depending on tier and number of units.
- Account for anomalies in sustained throughput due to data distribution, seasonality, and bursting.
- Use multiple cloud gateways when the IoT solution must manage millions of devices. Use DPS to assign devices to IoT hubs.

Provision devices with DPS

Use DPS to set up a connection to an IoT hub during provisioning, when the IoT Hub connection isn't available anymore, or during device reboot.

- Use the DPS evenly weighted distribution policy to adjust the weight for provisioning, based on use case. For more information, see How the allocation policy assigns devices to IoT Hubs.
- Consider provisioning devices to the IoT solution over a period of time, distributed or in smaller batches, to balance the DPS load and quota. When onboarding in batches, plan for the batches and overall migration timeline. Account for DPS limits in number of operations, device registrations, and maximum connections per minute, including latency and retries.
- Use DPS to allocate devices to IoT Hubs in different regions based on latency.
- Use a caching strategy for the DPS connection string to reduce DPS reconnect operations.

Manage downstream devices

An IoT solution is horizontally scalable if it has multiple gateways or edge devices per site or location and downstream devices that can connect to any of these gateways or edge devices.

- Use multiple gateways and edge devices in *translation* mode when the number of downstream devices, their messages and message sizes will change over time, and their protocol or message must be translated. Gateways and edge devices in translation mode can translate protocols or messages to and from downstream devices, however a mapping is needed to find the gateway a downstream device is connected to. Account for added message translation and buffering overhead at the gateway or edge device when you use translation mode.

- Use multiple gateways and edge devices in *transparent* mode to connect downstream Message Queue Telemetry Transport (MQTT) or AMQP devices when their number can change over time per site or location. Gateways and edge devices in transparent mode can connect MQTT/AMQP devices for bi-directional communication. Account for added message buffering, storage, and configuration overhead at the gateway or edge device when you use transparent mode.

Transport layer

The transport layer handles connections between a device and the IoT solution, transforming IoT messages to network packages and sending them over the physical network. IoT solutions commonly use AMQP and MQTT connection protocols.

Optimize resource usage

The connection between a device and the cloud needs to be secure, reliable, and scalable to handle the targeted number of devices and messages.

- Use an open stateful connection from a device to the cloud gateway. IoT Hub is optimized for managing millions of open stateful connections by using MQTT, AMQP, or WebSocket protocols. Keep open connections to the devices to minimize the overhead of security handshakes, authentication, and authorization. This practice improves performance and greatly reduces required bandwidth.

- Use an AMQP protocol that supports multiplexing multiple channels on a single connection to minimize the number of open connections the cloud gateway requires. By using multiplexing, a transparent gateway can connect multiple leaf devices using their own channels over a single connection.

- Use the device and module twins cloud gateway patterns to asynchronously exchange state information between devices and the cloud.

- Configure DPS to move the device state when a device connects to another cloud gateway.

Optimize data communication

The number and size of device to cloud messages affects performance and cost. Evaluating data communication is key to performance efficiency in your IoT workload.

- Use an efficient data format and encoding that doesn't use extensive bandwidth to send data to the cloud. For low bandwidth networks, consider using a compressed or binary format, but understand the overhead of uncompressing or converting the data in the cloud.
- Consider storing high-volume data locally and uploading it hourly or daily.
- Group many small device-to-cloud messages into fewer larger messages to reduce the total number. However, don't send only large messages, but balance between average message size and throughput.

Storage layer

The different types of data collected and referenced in an IoT solution often require storage types that are specialized and optimized for different scenarios on devices, gateways, and cloud. Data that must be available in multiple geographical regions globally or locally, and in some cases replicated to optimize latency, increases IoT storage complexity.

- Use a time-series database for storing time-series data that has timestamps and values. Enrich time-series data telemetry with columns for filtering, for example CustomerID, RoomID, or other use-case specific columns.
- Use device and gateway storage for caching data, or to keep data when disconnected. Account for required storage space. Don't keep all data, but use downsampling, store only aggregates, or store data for limited time periods.
- Consider separating data ingestion and event processing storage from reporting and integration storage needs.
- Use the data storage type that fits the need for required throughput, size, retention period, data volume, CRUD requirements, and regional replication. Some examples are Azure Data Lake Storage, Azure Data Explorer, Azure SQL, and Azure Cosmos DB.

Event processing and analytics layer

You can process data that devices generate before sending it to or within the IoT solution. Data processing can include translation, contextualization, filtering and routing, or more advanced analytics like trend analysis or anomaly detection.

Optimize edge versus cloud processing

Run real-time and near real-time workloads, or small, optimized, low-latency processing with time constraints, on devices or at the edge by using local compute. Run larger workloads, or other workloads that have added or external data, or compute dependencies, in the cloud.

For example, run a machine learning algorithm at the edge to count people in a video stream, and send an event containing the count to the cloud. Use the cloud to compare trends between different factories.

Run analytics workloads at the edge by using the Stream Analytics Edge module. For example, you can run anomaly detection at the edge and label the events sent to the cloud with the detected anomaly. When you run analytics at the edge, account for extra latency, late arrival, and windowing impact.

Be aware of the overhead of an edge workload with many connected downstream devices. The edge node must forward or process all messages and handle caching all the data if there's intermittent cloud connectivity. Validate the performance impact on your solution by testing with the planned maximum of downstream devices and messages per edge node. Be aware of the performance impact that message translation or enrichment can have on edge, IoT Hub, or cloud event processing.

Categorize individual workloads

Separate workloads by time constraint and required latency and response times, for example response within seconds vs. batch per hour. Hybrid hardware systems-on-a-chip (SoCs) can support workloads on the device level.

At the edge, use priority queues to separate different data streams with different priorities and TTL. For example, alarms should always be sent first but have a lower TTL than telemetry.

In the cloud, you can use consumer groups on Azure Event Hubs to separate out different data streams and handle and scale alarms differently from telemetry. You can also use IoT Hub routes to separate out different data streams, with filtering and separate endpoints. IoT Hub message routing adds some latency. Use Event Hubs, Azure Event Grid, or Azure Service Bus to distribute workloads while protecting against back pressure in the cloud.

Overly complex IoT Hub routing rules can affect throughput, especially routing rules with message body JSON filters, where every message needs to be deserialized and scanned.

Handle high-volume cloud data

To optimize performance efficiency for high-volume cloud data:

- Use out-of-the-box service integration between IoT Hub and data destinations like Azure Data Lake Storage and Azure Data Explorer that are already optimized for high performance throughput.
- Use the Event Hubs SDK to develop custom ingestion from an IoT hub with the included event processor. The event processor can rebalance devices and hosts.
- Use the right number of IoT Hub partitions and consumer groups for the number of simultaneous data readers and required throughput.
- Separate the storage needed for data ingestion and event processing from the storage needed for reporting and integration.
- Use the data storage that fits the needs based on required throughput, size, retention period, data volume, CRUD requirements, and regional replication. Examples are Azure Data Lake Storage, Azure Data Explorer, Azure SQL, or Azure Cosmos DB. For more information, see Select an Azure data store for your application.

Integration layer

The integration layer connects an IoT solution to other services and business applications.

- Separate the IoT solution ingestion pipeline from integration processing. Make sure complex queries or loads from the integration layer don't affect data ingestion performance.
- Use well-defined and versioned APIs for access to IoT data and commands.

- Avoid tools for end users to create user-defined queries against IoT data storage. Consider using separate data stores for integration and for reporting.

DevOps layer

Use the following DevOps mechanisms to maximize performance efficiency:

- A connected registry for local caching and deployment of container images.
- IoT Hub to update deployments to multiple devices at once, including devices and gateways.
- Device twins and module twins to update device configurations in a scalable and efficient way.
- Performance testing, including stress and load tests to replicate the production environment, such as location and heterogenous devices.

Monitoring

Use Azure Monitor to collect IoT Hub metrics with alerts for critical metrics. Set up Azure Monitor alerts based on current scale limits, such as device to cloud messages sent per second. Set the alert to a percentage of the limit, such as 75%, for pre-notification of upcoming scalability limits. Also set up Azure Monitor alerts for logs and metrics such as number of throttling errors.

Set Azure Service Health service alerts to trigger notifications when IoT Hub status changes.

Next steps

Reliability in your IoT workload

Related resources

- Performance efficiency principles
- Reference: IoT Hub endpoints
- IoT Hub message size
- Select an Azure data store for your application
- Monitoring Azure IoT Hub data reference
- Performance efficiency design principles

- Azure IoT reference architecture
- Azure IoT documentation

Oracle workload best practices

Article • 01/23/2023

Oracle workloads comprise not only Oracle Databases, but also applications such as Siebel, Peoplesoft, JD Edwards, E-Business Suite or customized WebLogic Server applications.

Customers around the globe are looking to host Oracle workloads on Azure. For that reason, Microsoft provides solutions for each workload. This includes guidance around network and reducing latencies, and also auto-scaling the solutions.

The section about Oracle workloads on Azure provides guidance that applies the Azure Well-Architected best practices as the technical foundation for building and operating a highly reliable solution on Azure at-scale.

About Oracle workloads

The term workload refers to a collection of database and application resources that support a common business goal or the execution of a common business process. These workloads could be Customer relationship management applications, Human Resources, customized application that mostly relies on WebLogic Server and others. WebLogic Server, Siebel, Peoplesoft, JD Edwards and E-Business Suite are Oracle on-premise applications.

Oracle applications on Azure and its approach are described in Application Design.

An Oracle Workload therefore describes a collection of application resources and databases, which must be highly available on the platform. The workload must always be resilient to failures.

What are the common challenges

Microsoft Azure makes it easy to deploy and manage cloud solutions. However, building and migrating Oracle workloads that are highly available and reliable on the platform remains a challenge for these main reasons:

- **Oracle applications architectures are complex**. The complexity relies on the dependencies between the application and database tier, Version upgrades and patches. Siebel has an architecture change between the version IP16 and IP17. The architecture change results in adjustments on the application and database tier to configure application servers on the database layer.

- **Designing a reliable application at scale requires knowledge about the application versions and architectures as well as dependencies on each layer**. By providing in-depth knowledge, we can help you to select the right technologies and guide you through the configuration setup.
- **Oracle database architectures differ from customer to customer**. Oracle RAC, Exadata features like SmartScan put complexity into the setup. Automatic Workload Repositories provide a great insight into the actual usage of Exadata Features or RAC Setup’s. Microsoft Azure provides solutions for every challenge so that we can address most of your needs.

All Oracle workloads need to be architected to handle failures with correlated or cascading impact. Reliability Engineering is an important task within the entire architecture design.

Assess the holistic architecture

Oracle workloads most often require high availability. High Availability is part of Reliability under Well-Architected Framework. High Availability has impact on the other pillars as well. If the architecture isn't assessed holistically, you might create unnecessary latencies. For that reason you should follow the recommendations of all pillars of the Well-Architected Framework.

- **Security**: The way user access applications and the holistic development of a workload architecture to mitigate security threats, such as Distributed Denial of Service (DDoS) attacks, will have a significant bearing on overall reliability.
- **Operational Excellence**: Potentially reached throughput or Memory on the database respond to operational issues will have a direct impact on application availability. Latencies should be eliminated to a minimum.
- **Performance Efficiency**: Availability is more than simple uptime, but rather a consistent level of application service and performance relative to a known healthy state.

Achieving high reliability imposes significant cost tradeoffs, which may not be justifiable for every workload scenario. It's therefore recommended that design decisions be driven by business requirements.

What are the key design areas?

Oracle on Azure guidance within this series is composed of architectural considerations and recommendations orientated around the below key design areas.

Design area	Summary
Application design	The use of a scale-unit architecture in the context of lift and shift highly reliable Oracle Applications. Also explores the Oracle application design patterns that allow for scaling, and error handling. Explore Oracle application dependencies between the web tier and application tier by understanding the shared connection between storage folders and databases.
Data Platform	Oracle on Azure offer different Virtual Machine sizes. The choice is based on the statistics of your Automatic Workload Repository (AWR) report for analyzing the current database usage. The AWR should always be handed in on peak-load.
Security	Oracle Applications have attached ports. It also depends on the operating system the application architecture has. Depending on the protocol, you can use different options to secure your access. Application Proxy, Azure Firewall, Application Gateway. Next to this you should take a look on the SSO method the version of the application has. It's important that we architecture your application according to best practices.
Networking & Connectivity	This section refers a bit to security as well. ExpressRoute helps you to set up a private connection to properly secure you and your environment. Microsoft Load Balancer should be configured to IP addresses (if necessary). Make sure that the network architecture doesn't create unnecessary latency.
Operational Procedures	Ansible or other Infrastructure-as-Code possibilities can help you to automate processes for Oracle workloads.
Health Modeling	Your Oracle workload environment should be properly maintained. Therefore checking the health status of the virtual machines and the ExpressRoute Connection should be used. On Oracle databases, you can use the Oracle Cloud Control.
Deployment & Testing	Before you upgrade applications, you should test it in your test environment. The same should be done on the Database side. Even when you use Infrastructure-as-Code you should test any deployment before putting in production.

We invite you to explore Oracle workload design best practices and return to this content regularly throughout the lifecycle of your Oracle workload. The content highlights critical areas of focus but also refers you to other documentation for deeper technical insight.

Application design

Oracle applications are complex. Especially the understanding of supported and unsupported functionalities from version to version make a lift and shift from version to version introduces difficulties in the migration approach. And if an understanding of each application version is missing most migration fails.

Moreover companies don't just want to lift and shift, but modernize the architecture and bind to functional and non-functional requirements. However, these requirements should be examined alongside key cloud application design patterns to ensure aspirations are fully achieved.

The design areas below explores both the subscription scale-unit approach and important application design patterns for building highly reliable Siebel, E-business Suite, JD Edwards, and Peoplesoft applications on Azure.

> ⓘ **Important**
>
> All Oracle applications are legacy systems and possess strong dependencies between the application tier and the database tier. Separating two tiers across different cloud vendors introduces latencies. Therefore you should do a proper technical assessment and decide if you are fine with it.

Oracle application designs

Oracle workloads such as Siebel, E-Business Suite, JD Edwards, and Peoplesoft have specific application design patterns for Azure. Below are example Azure architecture diagrams and more product detail of each Oracle workload designed for setting up internal and external user access. Note the application design considerations and the Azure services used for each design.

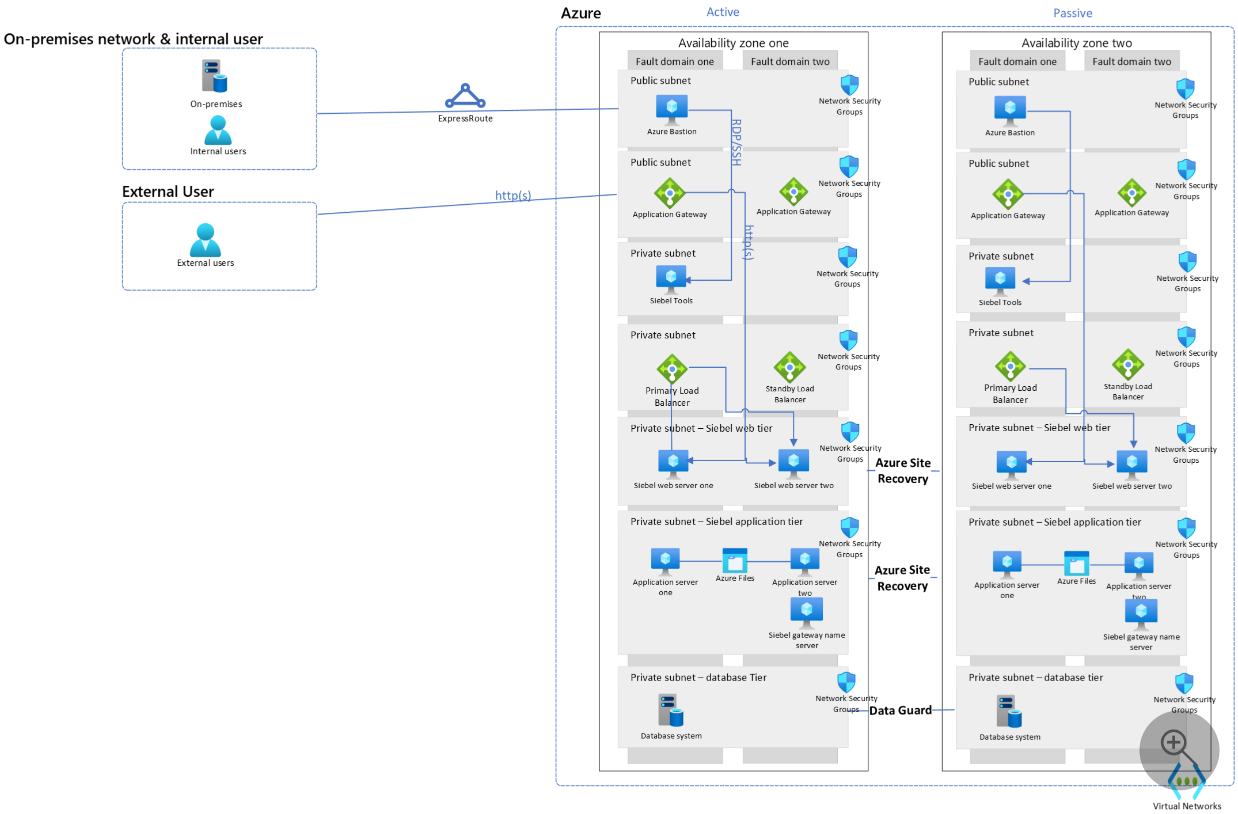
Siebel on Azure

Siebel is an on-premises CRM application that is used in many ways by companies. Siebel is one of the most complex applications in Oracle's portfolio.

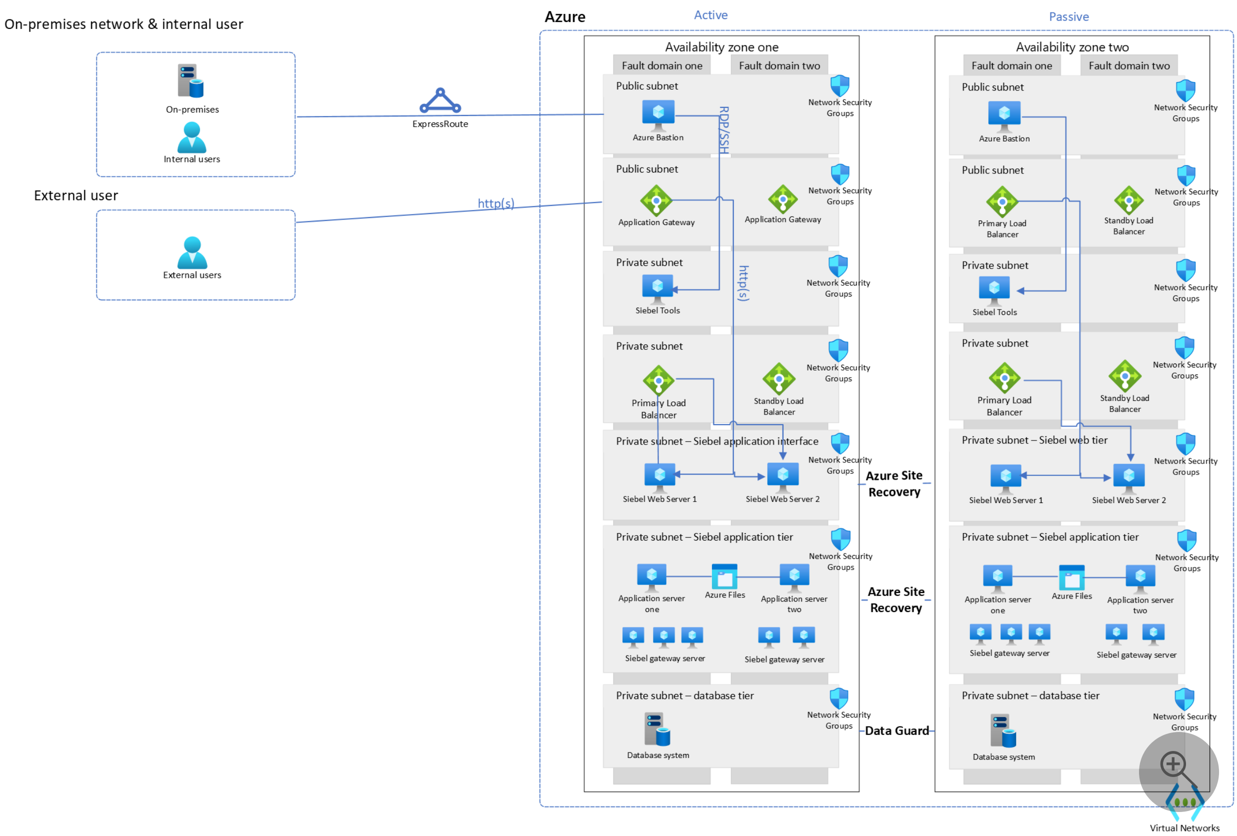
If you want to migrate your Siebel application, this must be done in the Siebel Tool subnet. Make sure that the database version matches up with the application version. Otherwise, this could be a first reason a migration fails.

In version 17 or newer, you need to configure certain servers and utilities on the application and database tier. As soon as the application tier is set up, all servers must be configured on the database as well. If all the tasks above haven't been conducted, either the migration will fail and/or the performance affected heavily.

The following design example provides a first idea of a generic Siebel Architecture on Azure for IP 16 and earlier:



The following design example provides a first idea of a generic Siebel Architecture on Azure for IP 17 and earlier:



Refer to the Siebel design considerations:

(1) Network & Security - You should consider establishing an Azure AD for Single-Sign-On (SSO). The SSO method used is Security Assertions Markup Language (SAML). Microsoft allocates an Azure AD Application Proxy. You should consider it especially for remote users. Internal users should be routed through ExpressRoute. If you desire a Firewall in front, Azure Firewall can be configured as well.

In cases where external users need to access your application, the Application Gateway provides a Web Application Firewall, but also a Layer 7 Load Balancing. Note that Application Gateways can only be used for http/s protocols. The subnets can be secured by using Network Security Groups (NSG). However if you desire to grant access only to certain individuals you can also use Role-Based-Access-Control (RBAC).

Because an SSH port is required for Siebel, a Bastion host as a jump box can also provide another security for an in-depth mature security posture.

(2) Application Web Tier - The web tier load balances the requests and send the requests accordingly to the application tier, database tier and/or backup.

(3) Application Tier - The application tier contains several tasks and servers. The web tier directs the requests into the application tier, shared storage system, and the database. The application tier has strong dependencies between the application server and the shared file system. The shared file System has a direct connection to the database. Cutting the dependencies causes latencies. A proper technical architecture assessment should take place before considering cutting these dependencies to different cloud vendors.

When you set up a holistic architecture and technical approach for migration you have the ability to establish it more modern. Microsoft offers various options. Auto-Scaling is interesting for companies that have events where they need to scale on high demands.

Virtual Machine Scale Sets can provide another value for Siebel. The Siebel Workflow Servers contain much information and for some customers it scales high and no need to redo architecture. Virtual Machine Scale Sets lets you create and manage a group of load-balanced VMs and provides high availability.

(4) Database Tier - The database tier has one primary and replicated to a secondary using Data Guard. If you stay within one datacenter, the synchronous configuration should be used. If you install your application across datacenters, you should configure Data Guard in Asynchronous mode.

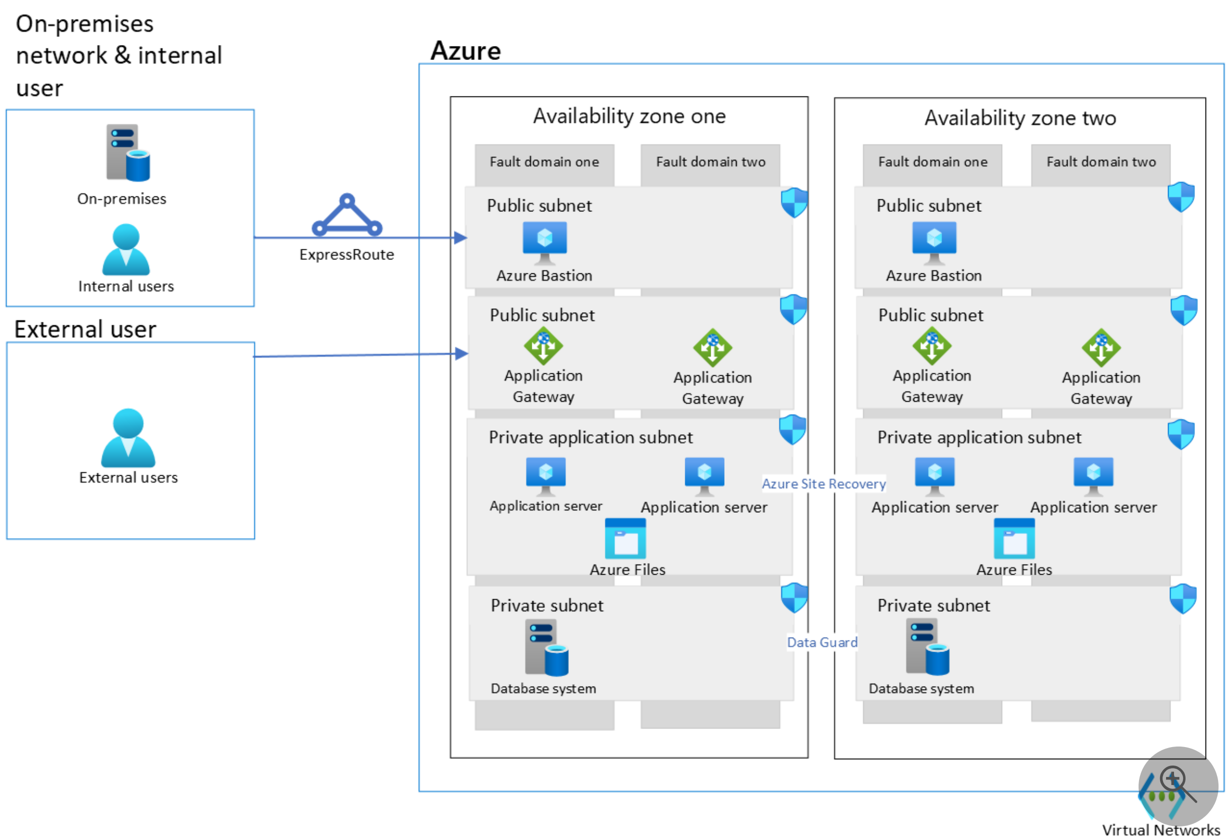
(5) Backup - Backups are sent from the application tier and the database tier. It's just one of many reasons why those two tiers shouldn't be separated into two different vendors. Thereby Backups of the database are performed by Azure Backup Volume Snapshot on Premium Files to the secondary region.

(6) Disaster Recovery - There are different solutions you can choose from. It very much depends on your requirements. The architecture above is built to be highly available. For replicating the application tier you can use Azure Site Recovery. Another solution you can choose is Redundancy options for managed disks. Both solutions replicate your data. Redundancy options for managed disks are a solution that can simplify the architecture but also comes with a few limitations.

E-Business Suite on Azure

Oracle E-Business Suite (EBS) is a suite of applications including Supply Chain Management (SCM) and Customer Relationship Management (CRM). As EBS is a SCM and CRM system, it usually has many interfaces to third-party systems. The below architecture is built to be highly available within one region.

We assume that external users don't cross the corporate network in the diagram below:



Refer to the Oracle EBS design considerations:

(1) Network & Security - You should consider establishing an Azure AD for Single-Sign-On (SSO). The SSO method used is Security Assertions Markup Language (SAML). Microsoft allocates an Azure AD Application Proxy. You should consider it especially for remote users. Internal users should be routed through ExpressRoute. If you desire a Firewall in front, Azure Firewall can be configured as well.

In cases where external users need to access your application the Application Gateway provides a Web Application Firewall, but also a Layer 7 Load Balancing. Note that Application Gateways can only be used for http/s protocols. The subnets can be secured by using Network Security Groups (NSG). However if you desire to grant access only to certain individuals you can also use Role-Based-Access-Control (RBAC).

Because an SSH port is required for EBS, a Bastion host as a jump box can also provide another security for an in-depth mature security posture.

(2) Application Tier - The application tier consists of more than two application servers. Usually, the rules for the application server are saved on application web servers. Evaluate the access you need to establish. If users only access the web tier but aren't allowed to access the application server, consider allocating the tiers into application web tier and application tier.

However, if the application tier and the database tier will be cut into two different cloud vendors, it can cause high latencies. We recommend you to do a proper technical assessment and decide if high latencies will impact your daily business.

As soon as you approach the technical assessment, you should consider modernizing it. Some customers ask for auto-scaling methods. Auto-Scaling can be achieved through Virtual Machine Scale Sets (Virtual Machine Scale Sets) for EBS. Some of the servers that you have in use operate more tasks and create a higher throughput than others. This should be evaluated during the Design Phase. Virtual Machine Scale Sets don't require a rearchitecture and let you create and manage a group of load-balanced VMs that provides high availability.

(3) Database Tier - The database tier has one primary and replicated to a secondary using Data Guard. If you stay within one datacenter, the synchronous configuration should be used. If you install your application across datacenters, you should configure Data Guard in Asynchronous mode.

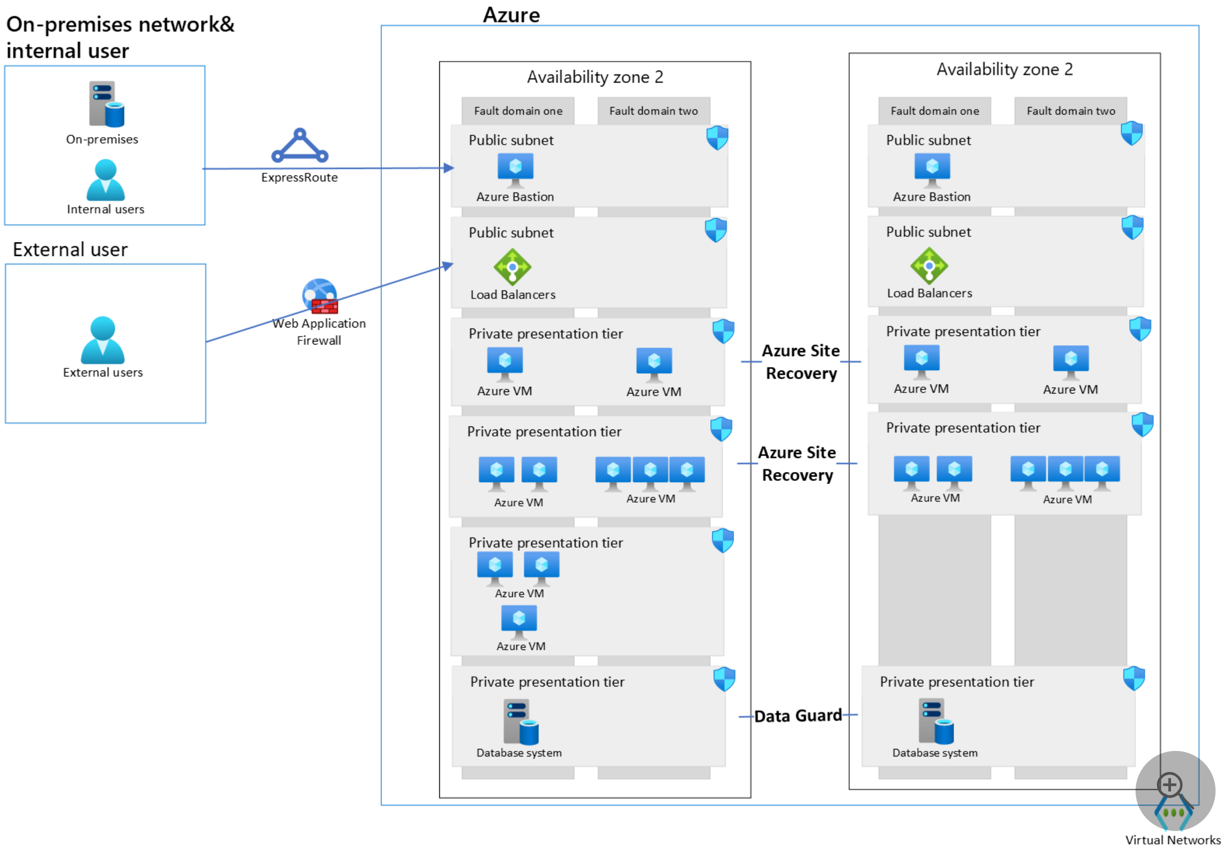
(4) Backup - Backups are sent from the application tier and the database tier. Thereby Backups of the database are performed by Azure Backup Volume Snapshot on Premium Files to the secondary region.

(5) Disaster Recovery - There are different solutions you can choose from. It very much depends on your requirements. The architecture above is built to be highly available. For replicating the application tier you can use Azure Site Recovery. Another solution you can choose is Redundancy options for managed disks. Both solutions replicate your data. Redundancy options for managed disks are a solution that can simplify the architecture but also comes with a few limitations.

JD Edwards on Azure

Oracle's JD Edwards is an integrated applications suite of comprehensive enterprise resource planning software. We have seen JDE used in Supply chain, Warehouse Management, Logistics, Manufacturing resource planning and more. Because of the use of the application, we see that interfaces to other systems are important as well.

The following architecture is built to being highly available. We assumed that external users aren't accessing over the corporate network. If an external user accesses the application using corporate network, the architecture can be simplified on networking.



Refer to the JD Edwards design considerations:

(1) Network & Security - You should consider establishing an Azure AD for Single-Sign-On (SSO). The SSO method used is Security Assertions Markup Language (SAML). Microsoft allocates an Azure AD Application Proxy. You should consider it especially for remote users. JDE does only offer an ssh port. In that case we recommend you to set a Firewall in front by using Azure Firewall to properly protect your application installation for external user.

Internal users can access your application through ExpressRoute. If desired, a Web Application Firewall on Azure Front Door can be configured as well.

The subnets can be secured by using Network Security Groups (NSG). However if you desire to grant access only to certain individuals you can also use Role-Based-Access-Control (RBAC).

Because an SSH port is required for JDE, a Bastion host as a jump box is recommended and can provide another security for an in-depth mature security posture.

(2) Application Web Tier - The application web tier consists of more than two application servers. Usually, the rules for the application server are saved on application web servers. Within the presentation tier, each instance is associated with storage. Cutting the dependencies can cause high latencies. Therefore we recommend to do a proper technical assessment.

As an often requested auto-scaling method you can explore Virtual Machine Scale Sets for JDE.

Some of the servers that you have in use operate more tasks and create a higher throughput than others. This throughput should be evaluated during the Design Phase. Virtual Machine Scale Sets don't require a rearchitecture and let you create and manage a group of load-balanced VMs which also high availability.

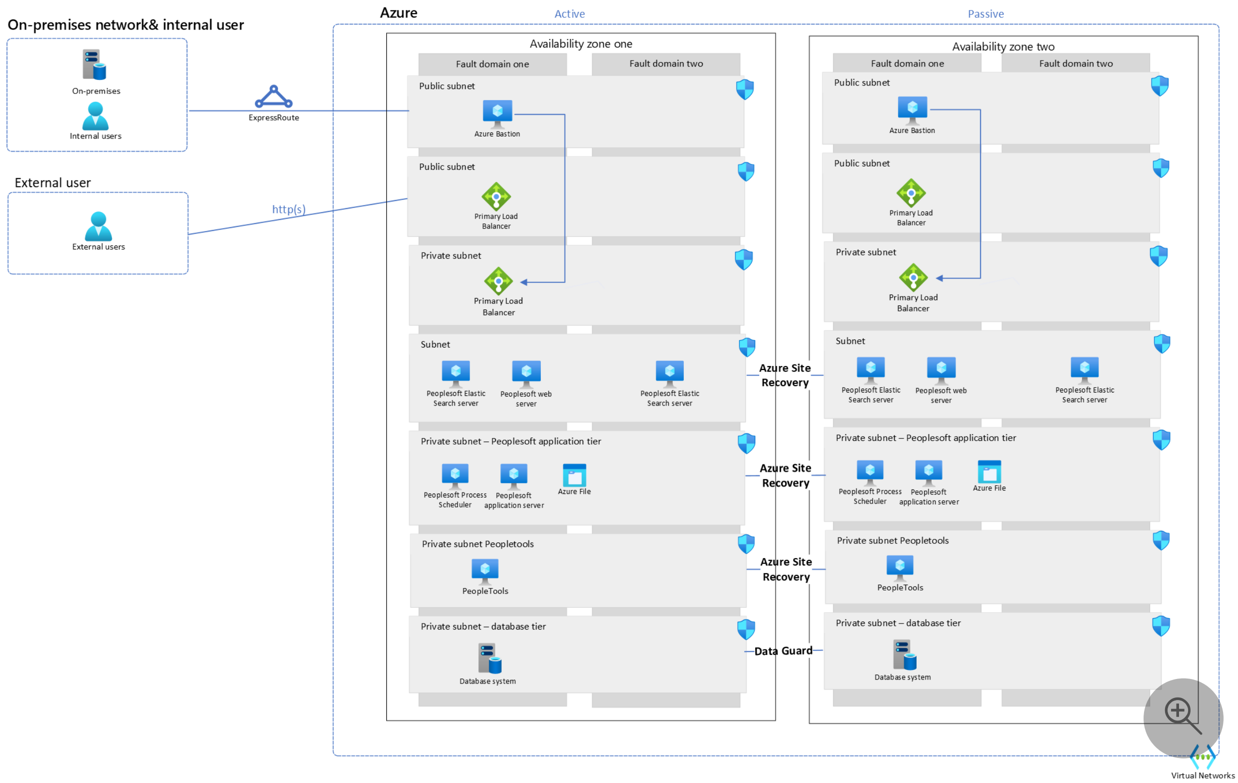
(3) Database Tier - The database tier has one primary and replicated to a secondary using Data Guard. If you stay within one datacenter, the synchronous configuration should be used. If you install your application across datacenters, you should configure Data Guard in Asynchronous mode. Data from the database tier are sent directly to an Azure Storage. The Storage is dependent on your current architecture setup.

(4) Backup - Backups are sent from the application tier and the database tier. It's just one of many reasons why those two tiers shouldn't be separated into two different vendors. Thereby Backups of the database are performed by Azure Backup Volume Snapshot on Premium Files to the secondary region.

(5) Disaster Recovery - There are different solutions you can choose from. It very much depends on your requirements. The architecture above is built to be highly available. For replicating the application tier you can use Azure Site Recovery. Another solution you can choose is Redundancy options for managed disks. Both solutions replicate your data. Redundancy options for managed disks are a solution that can simplify the architecture but also comes with a few limitations.

Peoplesoft on Azure

Oracle's PeopleSoft application suite contains software for human resources and financial management. The application suite is multi-tiered, and applications include human resource management systems (HRMS), customer relationship management (CRM), financials and supply chain management (FSCM), and enterprise performance management (EPM).



Refer to the Peoplesoft design considerations:

(1) Network & Security - You should consider establishing an Azure AD for Single-Sign-On (SSO). The SSO method used is Security Assertions Markup Language (SAML). Microsoft allocates a Azure AD Application Proxy. You should consider it especially for remote users.

Internal users should be routed through ExpressRoute. If you desire a firewall in front, you configure Azure Firewall.

In cases where external users need to access your application, the Application Gateway provides a Web Application Firewall, but also a Layer 7 Load Balancing. Note that the Application Gateway can only be used for http/s protocols.

The subnets can be secured by using Network Security Groups (NSG). However if you desire to grant access only to certain individuals you can also use Role-Based-Access-Control (RBAC).

Because an SSH port is required for Peoplesoft, a Bastion host as a jump box can provide extra security for an in-depth mature security posture.

> ⓘ **Important**
>
> Check the your operating systems for proper architecture and try to migrate as much to a Linux OS as possible.

(2) Application Web Tier - Requests entering the web tier are processed and sent to the application tier, database tier or Backup. The Elastic Search includes the indexes for the Peoplesoft application.

(3) Application Tier - The application tier contains several tasks and servers. It runs the business logic and processes but also maintains the connection to the database. As soon as this dependency is cut, it causes latencies. Therefore, the application and the database tier shouldn't be separated into two different vendors. The Process Scheduler is only required for Windows operating systems. For that reason, the architecture should be evaluated carefully.

Microsoft offers a couple of features for establishing a modern architecture. Auto-Scaling is interesting for companies that have events where they need to scale on high demands.

(4) Database Tier - The Database tier has one primary and replicated to a secondary using Data Guard. If you stay within one datacenter, the synchronous configuration should be used. If you install your application across datacenters, you should configure Data Guard in Asynchronous mode.

(5) Backup - Backups are sent from the application tier and the database tier. Thereby Backups of the database are performed by Azure Backup Volume Snapshot on Premium Files to the secondary region.

(6) Disaster Recovery - There are different solutions you can choose from. It very much depends on your requirements. The earlier mentioned architecture is built to be highly available. For replicating the application tier you can use Azure Site Recovery. Another solution you can choose is Redundancy options for managed disks. Both solutions replicate your data. Redundancy options for managed disks are a solution that can simplify the architecture but also comes with a few limitations.

Choose the right VM for your workloads

Every database and server is used to different capacities. Therefore, it is important to extract AWR reports on peak-load.

On the application tier, prepare numbers and statistics from the Web tier and the application tier. Statistics should include:

- Name of the server
- Number of CPUs
- Average utilization of CPU
- Memory size
- Average utilization
- App/ DB storage size
- App and DB version
- Operating system
- Total IOPS
- Total throughput
- Backup strategy

The E and M-series Virtual Machines are the right to choose from. Each Virtual Machine comes with a different size and can be perfectly matched to your needs. Microsoft also offers constrained VMs. These Virtual Machine are designed for customers not in need of many CPU but memory.

Recommendations for Oracle on Azure:

Recommendation	Benefit
Define functional and non-functional requirements	Includes the access of individuals, your backup strategy, RPO&RTO, availability, storage strategy to cost optimization, data retention, and security requirements
Right-Size your environment	Provide AWRs for your databases on peak-load and an overview about the statistics of the application server
Provide the version of your application	Different versions of Oracle Application have different limitations. You can help us to assess the right migration approach by just providing your application version
Security	Assess the version of your application for SSO methods. Some versions come with certain limitations
Availability	Microsoft offer a few availability methods of which you can make use of: Proximity Placement Groups, HA Storage types

Oracle and Microsoft have partnered to create a number of ready-to-deploy solutions for Oracle WebLogic Server on Azure IaaS. To get started, see What are solutions for running Oracle WebLogic Server on Azure Virtual Machines?.

Illustrative Examples

The Oracle workloads guidance provided within this series is based on a solution-orientated approach to illustrate key design considerations and recommendations. There are several reference implementations and scripts available.

For more information, see the Oracle reference architecture.

We invite you to explore Oracle workload design best practices and return to this content regularly throughout the lifecycle of your Oracle workload. The content highlights critical areas of focus but also refers you to other documentation for deeper technical insight.

For more information, see:

- Azure Center for Oracle Solutions
- Oracle workload in Azure
- Oracle workload architectures

Next Steps

Reliability

Oracle workload reliability

Article • 01/24/2023

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Use the following information to minimize down time and ensure that recommended practices for high availability are built into Azure and Oracle.

When discussing reliability with Oracle in Azure, it’s important to take into consideration not just the database, the connected tiers on separate VMs, virtual network subnets and disaster recovery if there's failures. Oracle on Azure IaaS can achieve these design considerations and have recommendations for each item.

Conduct a reliability assessment

Availability is crucial to Oracle Workloads. To discover the reliability status of your workload, you should answer some questions that help to identify areas of weaknesses. This is crucial for the architecture design and configurations.

The assessment provides specific recommendations to focus on. You can start the WAF assessment when prompted.

Create architecture reliability

Knowing the Architecture of your Oracle on-premises application is crucial. In some cases, specific versions of applications have a slight difference in the application tiers which are crucial for your migration phase. Next to it you should establish an API map pointing out any dependency. A dependency will most definitely occur between the application and database tier. The same applies to back up and your disaster recovery strategy.

Make sure to match up to the on-premises application architecture to reach high availability. It's most important for Siebel.

Make use of Azure Advisor

There are no specific built-in recommendations in Azure Advisor for Oracle on IaaS solutions in Azure. However, Azure Advisor has broad pillar coverage across the common resources used in these architectures, including compute (including Azure VMware), storage, and networking. Reviewing recommendations from Azure Advisor for these underlying resources can detect configurations & topologies that can lead to reliability impact, optimize cost from under-utilization of resources, and reveal opportunities improve user experience through surfacing throttling issues or proximity placement suggestions. See the related Azure Well-Architected service guide for each service in your architecture for a list of key Azure Advisor alerts for that service.

Create a fail-over in a multiple availability zone and second region deployments for disaster recovery

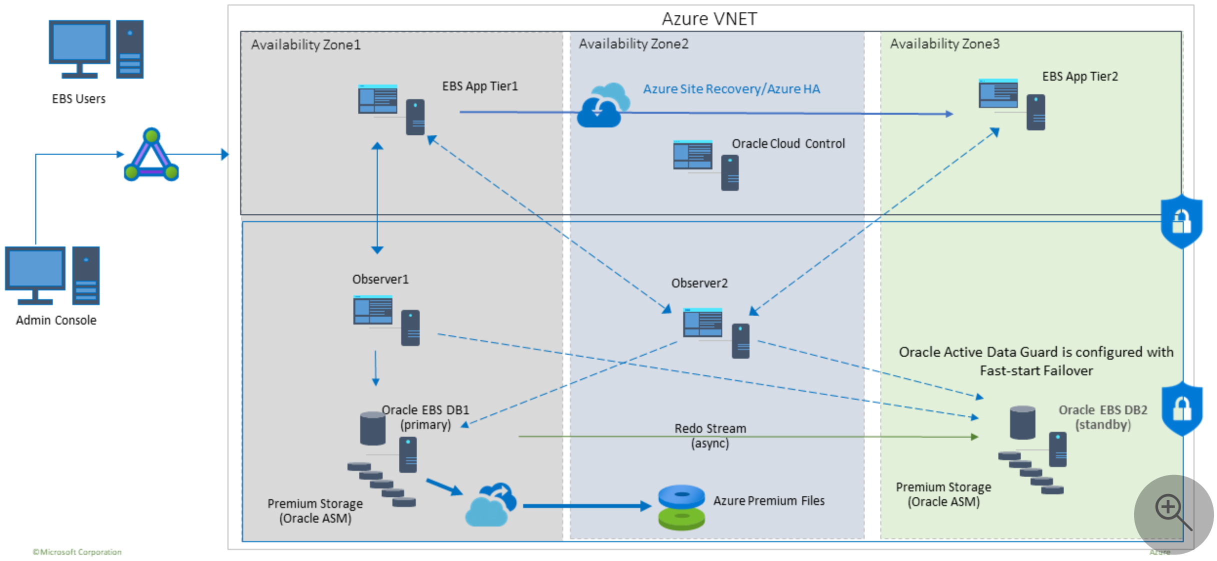
Business-critical Oracle Applications require failure prevention and therefore holistic architecture. One of these business-critical applications can be Oracle E-Business Suite.

As a given Tier 1 example for Oracle E-Business Suite in a multiple availability zone deployment and second region deployments for disaster recovery.

- First establish a multiple availability zone deployment with separated VNet with subnets. The Application tier uses Azure Site Recovery with a passive secondary virtual machine in availability zone three from the availability zone 1 primary.
- Use two Oracle Observers as a primary in availability zone one and a secondary in availability zone two. The observers monitor and direct the whole traffic to the primary database. Whereas the primary database is deployed in availability zone one. Oracle Data Guard performs the redo sync to availability zone two and can be configured for maximum availability. Data Guard can be established as synchronous or asynchronous. Within one region a synchronous configuration can be used for reaching a lower latency as in async mode.

A second Data Guard standby configuration in the secondary region is established for disaster recovery purposes and is configured for maximum protection. Thereby backups of the database are performed by Azure Backup Volume Snapshot on Premium Files to the secondary region.

If a primary goes down, Observer(s) will reroute the traffic to the secondary DB2 as it comes out of standby, becomes primary and takes over all functionality for environment and sequence to fail over to secondary region standby if regional outage in first region.

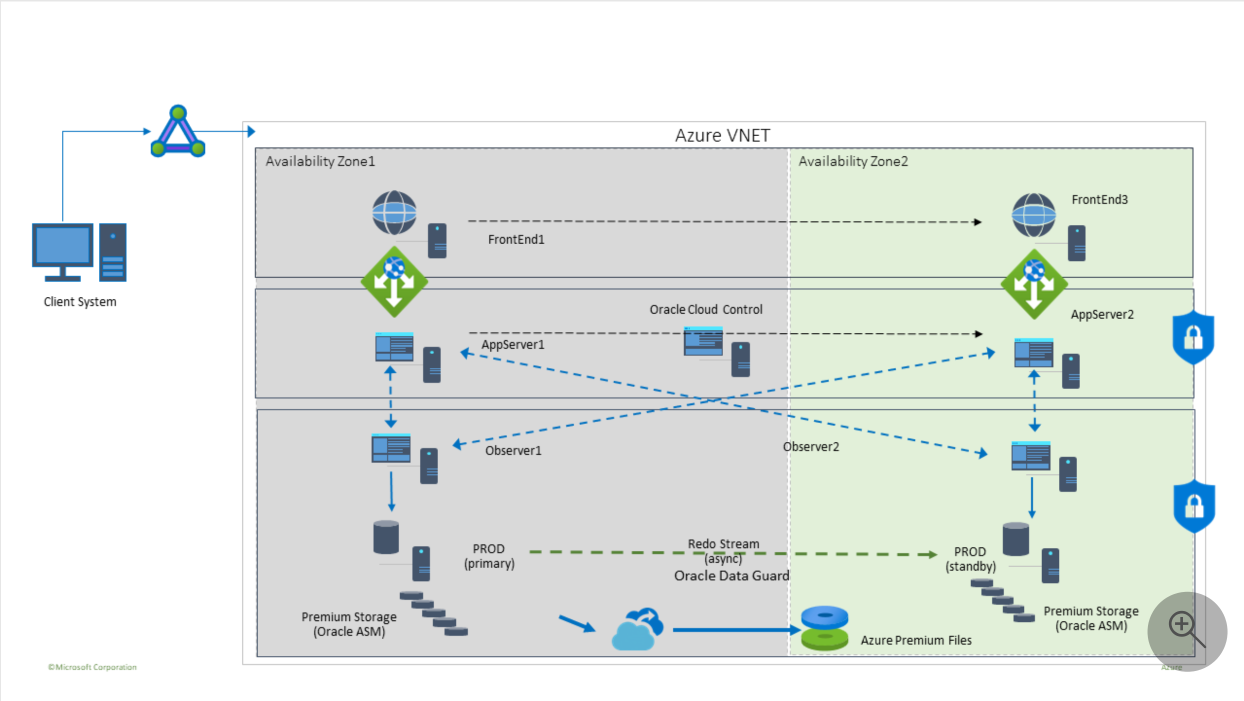


Create a failover for business critical Oracle applications in a two availability zone deployment with manual failover

The web server tier, application tier and database tier reside in its own virtual network subnet.

Azure Site Recovery or the manual clone utility can be established to duplicate the passive secondary in AZ2. The primary will be set up in availability zone one whereas the database uses Data Guard to replicate it to an active Standby in AZ2.

A failover would require manual intervention from the customer to fail over if there is a failure of availability zone one. Backups use Active Data Guard standby in AZ2 and backup to Azure Premium files in AZ2 to remove any additional IO pressure to the primary database.



Checklist for Reliability

The reliability checklist includes the Recovery Point Objective (RPO), Recovery Time Objective (RTO) and any business SLAs for the environment. The checklist should include all of the components for the environment, not just the database and may have different tiers of HA for different resources depending on their requirement for the business and system.

Choices for Oracle on Azure should include and be reviewed in the design principles for adding reliability to the architecture.

To assess the Utilization, extract the AWR or statspack workload data to right-size the database in Azure (lift and shift the workload, not the on-premises hardware.).

Microsoft offers different sizes of Virtual Machines matching CPU, Memory and throughput. Based on the data in the AWR or statspack you can choose the correct VM SKU for Oracle Workload to ensure reliability and sustainability of the Oracle workload.

Some explanations to the different types of Virtual Machines are:

- D-series newest version (v5 or higher) for small Oracle databases or non-production.
- E-series VM skus, newest version, (v5 or higher) whenever available for most production Oracle database workloads.
- M-series for high memory/vCPU use Oracle databases but be cognizant of IO limits for attached disk. Preference for network attached storage.

Choose a virtual machine (VM) based on the needs of vCPU, memory and IO threshold for storage attached to VM, dependent on attached managed disk or network attached storage.

Based on the AWR and statspack data you should also choose a Premium SSD for OS Disk and place Linux swapfile(s) on VM temp storage.

The appropriate storage for throughput (MBPs) for Oracle workload should be chosen.

Another solution when your workload requires a high throughput is Azure NetApp Files (ANF). For more product information, see Azure NetApp Files

For any multi-tier systems using E and D series VMs, consider using Proximity Placement Groups for more consistent performance between VMs.

Calculate the Recovery Point Object (RPO), Recovery Time Objective (RTO) and uptime SLAs for the system and ensure it matches for both the IT organization and the business.

Consider enabling host level caching on the VM and use readonly caching on premium SSD P30-P50 for Oracle Datafiles. Performances for read-heavy workloads require consistency for relational systems. Where bursting may offer some benefits, most often we rely on host-level caching for best and consistent Oracle workload performance. Read-only is chosen as Oracle desires all writes to write through to the datafiles to avoid hindering issues by caching.

Use accelerated networking for VM. To reduce the latency on the network, Azure ExpressRoute should be established.

The application health should be monitored, not just Oracle, but application and middleware tier (if existing):

- Oracle Enterprise Manager can be deployed on a separate VM to monitor, maintain, and manage Oracle environments.
- Oracle Enterprise Manager text logs can be loaded to Log Analytics and used with Azure Monitor for a single pane of glass monitoring.

Recommendations for your Oracle workloads on Azure

Recommendation	Benefit
Use Host-level ReadOnly Caching for Oracle datafiles	P30-P50, (up to 4095G) can offer almost double MBPs on reads for datafiles. Bursting isn't consistent or guaranteed, but caching offers better performance and at no extra cost.

Recommendation	Benefit
Use Host-Level ReadWrite caching on OS Disk	Oracle VMs can benefit with ReadWrite caching for the OS disk only.
Separate Redo Logs for Log Latency	Log File Sync and Log File parallel write waits in Oracle can benefit by separating log files too small ultra disk allocation. If sync waits are shown, and requirement to have two members of redo log groups, separate members to TWO ultra disks with high enough IO set to handle log latency. Don't configure any host-level caching if only redo logs on premium disk.

For more suggestions, see Principles of the reliability pillar.

Next step

Security

Oracle workload security

Article • 01/20/2023

Security is one of the most important aspects of any architecture. Azure provides all the tools needed to secure your Oracle workload. Oracle Applications can contain sensible data. Peoplesoft as an HR system is one of the examples that the whole architecture must be secured properly. This can be achieved through secured authentication methods, hardened networking, and encryption.

Oracle on Azure is delivered in the infrastructure as a service (IaaS) cloud model. We recommend you regularly evaluate the services and technologies used to ensure your security posture evolves with the threat landscape. Below are security recommendations for consideration. We recommend you review the security design principles.

Use identity management

Identity management is a framework that controls access to critical resources. Especially within Oracle applications lifecycle management is crucial. Part-time workers joining only during summer season, interns joining companies, or full-time employees. Many of these are in need of different accesses, which need to be checked and maintained, but also removed as soon as they leave the company. There are two identity management use cases to consider for your Oracle workload, and the identity management solution differs for each.

(1) Operating system - Operating system - Organizations can improve the security of Windows and Linux virtual machines in Azure by integrating with Azure Active Directory (Azure AD). Azure AD is a fully managed identity and access management service. For more information, see:

- Sign in to a Windows virtual machine in Azure by using Azure AD
- Sign in to a Linux virtual machine in Azure by using Azure AD and OpenSSH
- Sign in to a Windows virtual machine in Azure by using Azure AD

(2) Oracle application – Oracle Applications usually require a SSH (Port 22) and/or http(s) (Port 443) access. We recommend configuring single sign-on (SSO) using Azure Active Directory). SSO allows end users to connect to the Oracle applications via browser. For more information, see Azure Active Directory documentation.

The table below provides a summary of the recommended SSO method for the given Oracle solution.

Oracle solution	SSO methods
Siebel	Security Assertion Markup Language (SAML). From version IP18.1 and onwards: OAuth
Peoplesoft	From Version 8.53 and later: - Kerberos From Version 8.53 and later: - SAML - OAuth2.0 For more information, see Datawiza Azure AD
Hyperion	SAML 2.0
E-business suite (EBS)	SAML
JD Edwards (JDE)	SAML. For more information, see Datawiza JDE.

Microsoft also offers customers coming from on-premises application to use an application proxy. The application proxy allows SAML authentication and can be used for all Oracle applications providing you the opportunity to establish a Single-Sign-On for external users. For more information, see:

- Azure AD Application Proxy documentation.
- Tutorial on setting up Azure AD SSO Gateway for Oracle E-Business Suite - EBS, PeopleSoft, and JDE.

Use role-based access control (RBAC)

Role Based Access Control is a method to grant certain accesses to certain individuals. Azure RBAC is an authorization system build on the Azure Resource Manager. It provides a management layer to you that enables you to create, update, and delete resources according to the need of certain roles and individuals. For more information, see Azure RBAC documentation.

Enforce network and application security

Network and application security is crucial. Especially in regard to potential latencies, but also internet facing applications. Network security should be the baseline of every architecture.

(1) Azure Network Design - The first security option every customer has is a private connection to the cloud installation. In this case it's an MPLS (any-to-any networks) connection that is ExpressRoute. We recommend keeping the workload specific resources in one virtual network.

For Oracle-native setups, you should use Oracle Cloud Connector and Oracle Private Link for Azure as part of the hub-spoke setup. These technologies support the Oracle extension and innovation architecture for the Oracle Business Technology Platform (BTP). Azure native integrations fully integrated with Azure virtual networks and APIs and don’t require these components.

(2) Virtual network security - You can use Network Security Groups (NSG) to filter network traffic between Azure resources in your Azure virtual network. The NSG rules can be defined to allow or deny access to your Oracle application. It can also be dropped down to ports from on-premises IP address ranges. For more information, see network security groups.

(3) Application security - This is one of the most complex parts when it comes to Oracle applications. Determine where are users accessing the application and if they're only coming from the corporate network or also from the Internet. Take a look on the operating systems currently used whether Linux or Windows. Linux systems usually require access from SSH whereas Windows require an RDP protocol. Most customers desire to properly secure these protocols by installing a Bastion host in front. It's highly based on the customer requirements and also on where the application will be accessed.

Next to this it's also recommended to secure these accesses through a Web application Firewall. In case everyone logs in from the corporate network, this is only based on your security requirements and optional.

In case a user accesses the application from the internet, consider an Application Gateway. Azure Application Gateway provides two built-in functionalities. At first it operates as a Web Application Firewall, but also has a built-in Layer7 Load Balancer. It is only supported for access at Port 443 (http/s). For more information, see Azure Application Gateway

Another option to secure your network is the Azure Firewall. This component defends the web services against common exploits and vulnerabilities. It keeps the Oracle Application highly available and helps you meet compliance requirements.

Another possibility to configure a mature security posture are Application Security Groups (ASG). An ASG is an easier possibility to establish network security of the dedicated workload. For more information, see Azure application security groups.

Make use of Azure Policy

There are no specific built-in Azure Policy definitions for Oracle on IaaS in Azure. Azure Policy has broad coverage over the core resources that make up any Oracle solution on Azure. Primary resources are virtual machines, storage, and networking. Always enforce architectural choices with Azure Policy to prevent accidental drift from desired state. There are policies for these resources that span all five pillars of the Well-Architected Framework. If built-in policies for your architecture's components or configurations do not exist, you can create custom policies to cover any gaps.

Examples of key policies that are built-in would be Audit virtual machines without disaster recovery, virtual machines should use Secure Boot, and any policies that limit SKU choices to ensure you're meeting reliability, cost optimization, and performance efficiency targets. See the related Azure Well-Architected service guide for each service for a list of recommended Azure Policies for that service.

Recommendations

Explore the following table of recommendations to optimize your Oracle on Azure IaaS environment for security:

Recommendation	Benefit
Send STIG security alerts from Oracle Cloud Control to Activity Logs	Oracle Cloud Control STIG alerts and logs can be imported into Log Analytics to grant a single pane of glass for Cloud Management, while retaining Oracle tools for database specialists.
Use Listener Services	Follow recommended practices for the Oracle Net Listener to enforce local operating system authentication and manage incoming client connection requests to the database server.
Consider using the DBSAT to evaluate security state of database	Oracle provides a stand-alone command line tool called the Oracle Database Security Assessment Tool (DBSAT) to assess and check regulatory compliance, including relevant configuration and configuration information.
Virtual Machine Security in Azure	Follow the recommended practices for Virtual Machine Security in Azure, including the use of compute, data security, ports and network.

Next Steps

Cost Optimization

Oracle workload cost optimization

Article • 01/24/2023

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. We recommend you review the Cost optimization design principle.

Optimize workload compute costs

AWR and statspack statistics show your actual utilization. Based on your utilization, we help you to choose the right SKU to right-size according to your individual setup. Virtual Machines provide the computer power for your Oracle workload and have a cost effect that we can easily address. Below we provide some guidance to optimize your costs by choosing the right virtual machine type.

Choose the right VM type

Oracle workloads tend to be low utilized on the CPU but high on the memory. Next to the utilization of compute power and memory, it's important that the throughput fit into the VM as well. Addressing the fact that very low compute power is used, Microsoft offers you constrained VM sizes.

To match the Oracle workload into a Virtual Machine, we recommend to either use the E series or M series. Both E and M series sizes are memory optimized and able to handle a high throughput.

Optimize workload storage

All Oracle applications have a Shared File Storage. Data from the web tier but also from the database tier regularly sends the updated data into the File Storage. From there, the data is sent to back up storage. Storage can also have an effect of the performance, availability, and recoverability of your Oracle workload.

Set up a storage architecture that meets your requirements and allocating cost savings accordingly.

The following provides some guidance to set up a well-architected storage solution:

(1) Use reserved capacity storage type - There are several types of storage to choose from based on the requirements. Managed Disks, Blob Storage, and Azure Backup

Storage will support you in various combinations. As every application has a file storage in the application tier, a cost optimization can be reached by having a File Storage in Place. Network attached storage offers an opportunity for workloads with high throughputs. In this case it's Azure NetApp Files (ANF).

The above storage options come with storage reservation options that lower overall costs on the persistent data layer. For more information, please visit:

- Azure disk reserved capacity
- Azure NetApp Files
- Azure Backup Storage reserved capacity
- About Azure VM Backup

(2) Use lifecycle management policies - Next to reserved capacity you should define a data retention period. An Oracle database backup can be large and add cost to the storage costs if it isn't optimized. We recommend creating a lifecycle policy that meets the recovery time objective (RTO) and the recovery point objective (RPO) of your Oracle workload.

Recommendations

Explore the following table of recommendations to optimize your Oracle on Azure IaaS for Cost optimization:

Recommendation	Benefit
Familiarize yourself with right-sizing of Oracle databases	Based on our experiences most Oracle environments are over-provisioned. Learn how to take AWR or Oracle Statspack data and provide a right-size of the database(s) environment. Deliver expected performance without wasting resources.
Determine the most optimal pricing on storage that meets the workload requirements	Define the lifecycle management policy according to your requirements to the RTO and RPO. If the data reaches an age consider that the policy should move into Premium, Standard, Cold, Archive storage based on its age and business requirements.
Determine the IO and backup demands when using network attached storage	Some intense IO workloads might require an NFS. Other workloads might not require an NFS. There are situations where sizing and IO will result with higher tiers of storage having lower cost.
Monitor costs and create budget alerts	The Azure portal equips you with various options and possibilities to monitor the costs. Alerts will help you.

Recommendation	Benefit
Before scaling up hardware, ensure you understand what is causing performance bottlenecks and if it’s at the database or hardware level	Optimizing workloads can save a considerable amount over time and potentially also CPUs that you actually don't need. Make sure to regularly take a look on the AWR reports. When there are performance issues, it can also be caused on a storage level. Make sure to understand issues first before scaling up.
Familiarize yourself with the Oracle licensing FAQ	Be aware of what products are supported in Azure and how licensing works in Azure as a supported cloud for Oracle.

Next Steps

Operational Excellence

Oracle workload operational excellence

Article • 03/10/2023

Operational excellence is about creating efficient processes to support your Oracle workload. Operations will be the longest phase of the Oracle workload lifecycle, and teams must be equipped with operational best practices to manage the day-today tasks. Failure in operations will affect the other design areas and the overall success of the Oracle workload. It's critical to tailor your operations to support an Oracle workload in operations. Below are recommendations to drive operational excellence.

Monitor the workload

Monitoring and diagnostics are crucial for Oracle on Azure IaaS. As Oracle is a non-native solution running in Azure, Oracle Cloud Control offers the ability to monitor, manage and automate much of Oracle with little additional cost. Oracle Cloud Control comes with a Limited Use Enterprise Edition License, so the database repository does NOT require additional licensing, only the management packs are licensed to each target database. This ensures the same or similar tools that Oracle technologists have used on-premises are available in the cloud. We recommend you review the Operational excellence design principles.

Oracle Cloud Control has text logs with a clear format that can be exported and loaded into Azure Log Analytics to offer a single pane of glass when monitoring Oracle database information. Cloud Control can be used with management packs to add more functionality, including "simulating" a PaaS experience for IaaS solutions such as Oracle in Azure. When using the Cloud and Lifecycle Management Packs for Oracle Cloud Control, full cycle management can be added to the system- automating patching of both database and OS, along with cloning and deployments.

Azure policy and operational excellence

Many built-in Azure Policy definitions come in Deploy if Not Exists (DINE) versions for the resources that commonly make up Oracle solutions on Azure. Infrastructure as code (IaC) is the gold standard for resource deployment in business-critical applications, such as Oracle workloads. DINE policies perform imperative operations on your infrastructure and happen outside of the declarative model of IaC. Consider their usage carefully, opting for IaC-based deployments of infrastructure and architecture decisions, leaving Azure Policy for governance, not workload deployment/configuration.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for Operational excellence:

Recommendation	Benefit
Use Perf Insights	Use Perf Insights to monitor the overall cloud landscape.
Configure ExpressRoute connection monitoring between your on-premises and Azure network.	In case of failures or performance issues, it provides a first view into the network.
Consider Oracle Cloud Control for Oracle	Oracle is a non-native product and Oracle provides an infrastructure monitoring, management and automation tool that can be deployed on a VM with little to no cost to the customer, as it's most likely already used on-premises. Deploy to the cloud to eliminate egress to an on-premises version and evolve to automate with: - Monitoring templates - Metric Extensions (self-service metrics) - Corrective Actions - OS and database patch automation Oracle Cloud Control will "simulate" a PaaS like solution for IaaS when using it to manage, monitor and automate Oracle.

Next Steps

Performance Efficiency

Oracle workload performance efficiency

Article • 03/10/2023

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. We recommend you review the Performance efficiency principles.

Oracle on Azure IaaS has components that are distributed across various Azure services. These components are the capacity planning targets for your application servers and database sizing choices. This includes load balancing configurations for even traffic distribution and Oracle-native solutions, like Oracle Data Guard for fast-start failover database setups.

Optimize workload compute

Compute includes the hardware, number of cores, and memory but also the throughput. It's important to know the demands of your workload. Therefore, regularly look on the AWR reports and statistics. It shows the actual needs of your workload so that you can right-size your environment. Where possible and required you can add auto-scaling on the VMs. It can potentially be achieved through Virtual Machine Scale Sets.

Optimize workload storage

By having the correct solution, you gain the ability to improve the performance of the existing capabilities and allow yourself to add new features. In general the storage should meet the input output operation per second (IOPS) but also the functionalities. Microsoft offers different solutions that can simplify the architecture.

Optimize workload networking

Oracle applications usually have a high number of interfaces to other systems. Common communication paths are local storage, external storage, VMs in the network, and third-party applications. Optimizing the workload network with Proximity Placement Groups can bring higher performance.

Understanding Proximity Placement Groups

A proximity placement group is a logical grouping. It makes sure that Azure compute resources are physically located close to each other. Proximity placement groups are useful for workloads where low latency is a requirement, which is typical for Oracle Workloads.

Proximity placement groups reduce the distance between Oracle workloads. They can group different VM types under a single network spine. As the Azure footprint grows, a single availability zone may span multiple physical data centers. The distribution across data centers can create network latency impacting your Oracle application performance. The proximity of the VMs provides lower network latency.

The capabilities are ideal, but there are drawbacks to be aware of. Proximity placement groups often limit your VM choices and make resizing VMs more difficult. Proximity placement groups bind VMs to a specific network spine. This binding limits the possible combinations of different VM types. The host hardware that is needed to run a certain VM type might not be present in the data center or under the network spine to which the proximity placement group was assigned. The availability of VM types can be severely restricted.

We recommend using proximity placement groups in two scenarios:

1. Use proximity placement groups in Azure regions where latency across zones is higher than recommended for the Oracle workload.
2. Use proximity placement groups for application volume group. The Application Volume Group feature of Azure NetApp Files (ANF) uses PPG to deploy ANF volumes close to the VM/compute cluster. We recommend using this feature as designed.

For more information, see:

- Overview of proximity placement groups

Recommendations

Explore the following table of recommendations to optimize your Oracle database environments for performance efficiency:

Recommendation	Benefit
Move from RMAN backups to Volume Snapshots	Databases larger than 1 TB can benefit from removal of streaming backup utilities, such as RMAN or Datapump (exports) and can save 20-40% IO used by these utilities.

Recommendation	Benefit
Collect size of database, redo generation, and backup retention/size to determine the target minimal load on resources and the network.	AWR reports provide an insight into it. The numbers will help you to right-size your environment.
Determine if using premium SSD is best for the OS disk with the earlier mentioned sizing assessment in mind.	If using premium, enable host level read-only caching for datafiles.
Ensure host level caching is disabled for redo logs.	Higher Performance.
Test the VM chosen that the workload it was sized to support meets the needs once running in Azure.	Include implementing load balancing and autodetection so that as services are added or moved, the application can perform the necessary routing – it prevents you reaching high latencies.
Increase the size of the ExpressRoute gateway.	Upgrade to a higher gateway SKU for improved throughput performance between on-premises and Azure environment.
Move the right database files to higher IO storage	Use the AWR/Statspack report on IO Summary by file type to ensure that what you offer more IO to, is what will benefit from faster storage.

Next Steps

Oracle Reference Architecture

SAP workload documentation

Learn how to build an SAP workload on Azure

Get started

OVERVIEW

SAP workload

Design principles

Design areas

CONCEPT

Application design

Application platform

Data platform

Networking and connectivity

Security

Operational procedures

Reference examples

ARCHITECTURE

SAP on Azure landing zone accelerator

Start small and expand with SAP HANA

Reference implementations

DEPLOY

SAP on Azure Deployment Automation Framework

Learn

TRAINING

How to use SAP on Azure Solutions

Assessment

HOW-TO GUIDE

Azure Well-Architected Review (SAP on Azure)

SAP workload

Article • 01/09/2023

SAP is one of the world's leading producers of software solutions for business management and customer operations. SAP provides a suite of powerful applications that you can configure to meet specific environment and organizational needs. These applications facilitate data processing and information flow across organizations and provide critical capabilities that drive key organizational functions.

SAP applications thrive on infrastructure that is tailored to maximize their capabilities and with cloud services designed to optimize SAP workload functionality. Microsoft has hosted SAP instances in its data centers for decades and has custom built infrastructure and services to run SAP and its applications as they were designed. We've used this SAP experience to create a concise set of best practices for evaluating, designing, and optimizing an SAP workload from premigration to operations. The goal of this guidance is to help you get the most benefits from each SAP workload.

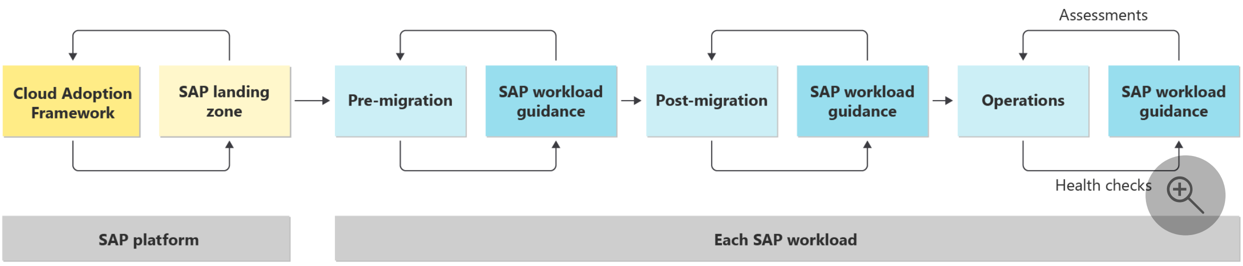


Figure 1: Using SAP workload guidance throughout the lifecycle of an SAP workload.

The guidance is designed for one SAP workload at a time. Anytime you want to optimize a specific SAP application (S4/HANA, NetWeaver, etc.) for Azure or in Azure, you should use this guidance. Work through the guidance as many times as you need to derive the benefits you expect. In operations, this guidance should be paired with the Well-Architected Review assessments and health checks. We address these tools in our guidance.

Start with SAP platform landing zone. Before using this content, you should have an SAP platform landing zone in Azure. The platform landing zone provides shared services to one or more of your SAP workloads. If you don't have a platform landing zone, you should use the SAP cloud adoption framework and deploy the SAP landing zone accelerator. The landing zone accelerator establishes the required foundation for your SAP workload.

For more information, see:

- SAP landing zone accelerator

- SAP cloud adoption framework
- Azure landing zone overview

We invite you to explore SAP workload design best practices and return to this content regularly throughout the lifecycle of your SAP workload. The content highlights critical areas of focus but also refers you to other documentation for deeper technical insight.

For more information, see:

- Azure Center for SAP Solutions
- SAP workload in Azure
- SAP workload architectures

Next steps

Our guidance adheres to five pillars of the Well-Architected Framework. It's important to understand what those pillars mean for an SAP workload and how they interact at a high level.

Design principles

Design principles of an SAP workload

Article • 01/09/2023

We built this guidance around the Azure Well-Architected Framework and its five pillars of architectural excellence. The table below lists each pillar and provides a general summary of the articles in this set.

Well-architected framework pillar	Summary
Reliability	An SAP workload requires resiliency at the architecture layer. You'll learn how to create an SAP application with high availability to process critical business data.
Security	An SAP workload contains critical business data. You'll learn to secure your SAP applications with multiple security layers, including identity, access, input validation, data sovereignty, and encryption.
Cost Optimization	An SAP workload has several architecture layers and numerous resources supporting it. You'll learn how to make sure your SAP application deployment meets performance expectations while reducing the total cost of ownership.
Performance Efficiency	An SAP workload needs to be high performing resources to meet productivity requirements. You'll learn how to make sure that your SAP workload meets user demands while managing costs.
Operational Excellence	An SAP workload spends most of its lifecycle in operations. You'll learn how to manage an SAP workload and to keep it running.

Reliability

A reliable SAP workload is both resilient and available. Resiliency is the ability to recover from failures and continue to function. Availability is uptime. High availability reduces SAP application downtime during critical maintenance and improves recovery from failures such as VM crashes, backend updates, major downtime, or ransomware incidents. Failures happen on-premises and in the cloud, so it's important to design your SAP workload for resiliency and availability.

Conduct a reliability assessment. Before you can standardize the reliability of an SAP workload and improve areas of weakness, you need to assess its reliability. It's critical to know how reliable an SAP workload is so steps can be taken to fix issues or solidify those configurations. We recommend conducting a reliability assessment on your SAP workload. The assessment asks you questions about your workload and provides specific recommendations to focus on. The assessment builds on itself, so you can track your progress without restarting every time.

For the assessment, start an Azure Well-Architected Review. Select "Start Assessment" and “SAP on Azure” when prompted.

Security

SAP on Azure is delivered in the infrastructure as a service (IaaS) cloud model. Microsoft builds security protections into the service at the levels of the physical data center, physical network, physical host, and hypervisor. But you're responsible for areas above the hypervisor, such as the guest operating system for SAP. We recommend you regularly evaluate the services and technologies used to ensure your security posture evolves with the threat landscape.

Cost optimization

Microsoft makes significant investments in the fast evolution of its SAP hardware and services to provide more value for less. The frequent increase in Azure hardware capability provides regular opportunity for an SAP workload to optimize costs, eliminate waste, and improve technology. To align Azure and your SAP workload, we recommend creating a plan for each SAP workload. The plan should contain the objectives and motivations for the workload. Organizational objectives and investment priorities should drive cost optimization initiatives for your application, application platform, and data platform.

Performance efficiency

Performance efficiency is about accelerating digital transformation with less. The goal is to get the most out of your SAP workload and meet user demand without over or under provisioning resources. Inefficient performance can degrade user experience and inflate costs. Performance affects productivity for internal applications. It determines growth for public facing applications. Designing an SAP workload that can't meet user demand will slow the application. Overcompensating with too much compute power will drive up cost needlessly. These scenarios are avoidable with the right compute, data, and networking design.

Operational excellence

Operational excellence is about creating efficient processes to support your SAP workload. Operations will be the longest phase of the SAP workload lifecycle, and teams must be equipped with operational best practices to manage the day-today tasks. Failure in operations will affect the other design areas and the overall success of the SAP workload. It's critical to tailor your operations to support an SAP workload in operations. Regular assessments, monitoring, and automation are central to improving SAP operational excellence.

Next steps

We've woven these design principles throughout our guidance in specific design areas. The design areas provide targeted guidance. We want you to find what you need fast so you can be more productive with less time. Use the headings as a compass to find appropriate direction in each design area.

Application design

Application platform

Data platform

Networking and connectivity

Security

Operational procedures

SAP workload application design

Article • 01/09/2023

SAP applications should adhere to the design principles. The guidance here focuses on cost optimization and reliability.

SAP application cost optimization

Optimizing your SAP application can lower the total cost of ownership without reducing capabilities. The goal is to generate the maximum return on investment (ROI). Here are ways to optimize an SAP application.

Identify application responsibility. Optimizing an SAP Application should be the responsibility of the customer business application team. Having someone or a group responsible for costs will help drive decisions that optimize costs over the lifecycle of the SAP workload.

Rationalize and rearchitect. You should consider rationalizing or rearchitecting the SAP application, especially during migrations. S4 HANA often replaces older SAP applications that can be added as a legacy system. The SAP WAF assessment can help validate rearchitecting efforts and should be conducted on a periodic basis. For more information, see Azure Well-Architected Review.

Minimize investment in legacy systems. You should host a legacy SAP application on minimum-supported architecture to help reduce cost. A legacy application is slower and less performant. Any legacy systems that remain after rationalizing and rearchitecting should receive the minimum spend possible and be retired when appropriate. For more information, see Azure Cost Management.

SAP application reliability

Use a multi-tier architecture. Creating a multi-tier architecture to support an SAP workload is essential for reliability. The number of tiers and architecture varies for each SAP application. Make sure to isolate application components from each other and create redundancy to achieve high availability. Where applicable, you should isolate the SAP Web Dispatcher, SAP Central Services, SAP App Server, SAPMNT Share and database. We have sample architectures for several different SAP applications you can use to inform your design.

For more information, see:

- SAP S/4HANA in Linux
- SAP BW/4HANA
- SAP NetWeaver

Configure SAP central services reliability. SAP central services (SCS) or ABAP SAP central services (ASCS) is the basis of SAP application communication. It consists of the message server and enqueue server. The central services layer is often a single point of failure and must be set up for high availability to achieve SAP application resiliency. To add redundancy, create a cluster of SAP central services with compatible shared storage technology supporting the cluster. Depending on the operating system and available shared storage technology in general availability or private/public preview, various options are available. Availability zones provide an opportunity to create a highly available ASCS architecture.

For more information, see:

- SAP workload configurations with Azure Availability Zones
- High-availability architecture for an SAP ASCS/SCS instance on Windows
- High-availability architecture for an SAP ASCS/SCS instance on Linux

Next steps

Application platform

Data platform

Networking and connectivity

Security

Operational procedures

SAP workload application platform

Article • 01/09/2023

The application platform refers to the hosting environment, application dependencies, frameworks, and libraries. For an SAP workload, the Azure platform provides opportunities to optimize cost and performance, allowing you to do more with less.

Optimize compute costs

Compute cost optimization is achieved through planning, monitoring, and resizing VMs throughout the SAP workload lifecycle. VMs provide the compute power for the SAP application and have a direct effect on cost and performance. We recommend monitoring the compute costs of an SAP workload to ensure the dollars spent are helping you meet organizational goals. Here are cost-optimization recommendations for SAP workload compute.

Choose the right VM type

Azure has SAP-certified VMs for your workload. The wrong VM type will require larger sizes to get the performance need, increasing cost without benefit. A smaller VM of the correct type can give you equal or better performance than a large instance of the wrong type. Azure offers a list of SAP-certified configurations to help you understand what VMs work well with your business needs.

For more information, see SAP certified infrastructure .

Memory-optimized VMs can meet the requirements of most SAP applications. An SAP online analytical processing (OLAP) workload and an online transactional processing (OLTP) workload should use M-series VMs. Examples of an OLTP workload include SAP HANA, SAP Business Suite on HANA, and SAP S/4HANA. Examples of OLAP workloads include SAP BW on HANA BW/4HANA. SAP Business One on HANA pairs with M-series VMs.

SAP NetWeaver, Business All-in-One, Business Suite Software, and BusinessObjects BI have broader alignment with different VM types. They can run on VMs in the D, G, E, and M-series.

Optimize compute cost during migration

Many SAP journeys start on-premises, so it's important to plan for compute optimization throughout the migration of a workload. Make sure to follow best practices across every SAP migration. You can find process guidance in our CAF SAP documentation. With the broader process guidance in place, you'll still need to customize your compute optimization for each SAP workload. We want you to consider pre-migration and post-migration milestones.

Optimize pre-migration. Pre-migration optimization ensures you have sufficient cloud resources provisioned to support the expected migration runtime of the SAP workload. You need to verify that the Azure VM meets the technical requirements of the on-premises workload. Planning will shorten the migration time of a workload and minimize the time of migration will keep costs lower.

Optimize post-migration. Post-migration optimization focuses on the end-user experience. This step coincides with the hypercare period, a time of elevated customer service to make sure that the workload is performing. You should monitor the workload as users begin to interact with it. The performance metrics might indicate that you need to downsize the VM or switch to a different VM type.

Improve application platform operational excellence

It's important to optimize VMs in operations for the most cost-savings. By VM operations, we're referring to the daily management of an SAP workload. This phase of a workload brings the ability to predict the compute needs. It's important to see how user demand affects compute needs over time. The VM choice should change along with the SAP workload's requirements. Here are cost-saving recommendations for operations.

Use Azure Advisor. We recommend you use Azure Advisor to identify VM usage that needs optimization. For more information, see Azure Advisor cost optimization.

Enforce VM governance. You should enforce VM governance at VM creation as a cost and security best practice. Every VM deployment increases the operational cost and attack surface of the SAP workload. VMs created outside of the governance process tend to create unneeded expense and have more vulnerabilities. We recommend using Azure Policy to enforce VM governance for your SAP workload. For more information, see Azure Policy for SAP.

Stop non-critical workloads. Each SAP workload has different levels of criticality. Some workloads aren't needed on nights and weekends. A sandbox environment is a good example of low criticality. We recommend stopping VMs that support non-critical workload environment to reduce costs. You can automate this process for the SAP workload in Azure. For more information, see automate VM start and stop.

Use Reserved Instances. Any SAP workload that needs to run continuously should use reserved instances to optimize cost. For budget predictability, you can make an advanced purchase for one or three years in a specified region. For more information, see Azure Reservations.

Use the Azure Hybrid Benefit. Azure lets you use on-premises Software Assurance-enabled Windows Server and SQL Server licenses. The benefit applies to Red Hat and SUSE Linux subscriptions. This benefit can generate significant savings for a hybrid SAP workload. For more information, see hybrid licensing benefit.

For more information, see:

- SAP on Azure
- SAP NetWeaver
- SAP HANA install
- SAP HANA configuration

Configure application sever reliability

The goal of application server reliability is to have multiple application servers load balance traffic and failover when needed. Resiliency for the SAP application server layer can be achieved through redundancy. You can configure multiple dialog instances on different instances of Azure virtual machines with a minimum of two application servers. Here are application server resiliency recommendations.

Use Availability Sets / Availability Zones. An SAP application server can be deployed in an availability set or across availability zones. The decision you make needs to be based on workload requirements. We recommend you choose one method to improve resiliency, but we don't recommend scale sets. For more information, see availability zones for SAP.

Use multiple application servers. Using multiple smaller application servers instead of one larger application server is recommended. This setup avoids a single point of failure. It's a best practice to configure SAP Logon Group (SMLG) and Batch Server Group (RZ12) for better load balancing between end-user & batch processing.

For more information, see:

- Azure Virtual Machines high availability for SAP NetWeaver
- SAP HANA high availability for Azure virtual machines

- SAP workload configurations with Azure Availability Zones

Improve compute performance efficiency

Compute is the core that powers an SAP application. Compute includes the hardware, number of cores, and memory. These features are foundational to organizations. If you don't optimize your compute configuration, an SAP workload will be unable to meet spikes in user demand or stay withing predefined budgets. It's important to know the demands on your workload and match those demands with the compute you use for your SAP workload. Here are some compute performance considerations.

Conduct reference sizing for on-premises workload. Reference sizing is the process of checking the configurations and resource utilization data of an SAP workload on-premises. Reference sizing data shows the current compute needs of the workload, and these needs should be matched in Azure. To find this information, use the SAP OS Collector. SAP OS Collector retrieves system utilization information that can be reported via SAP transaction OS07N and the EarlyWatch Alert. Any system performance and statistics gathering tools can collect similar information.

Use SAP Quick Sizer for a new workload. SAP Quick Sizer is a free web-based tool developed by SAP that translates business requirements into technical requirements. Use this tool when you build a new SAP workload to find the Azure VM with the correct network and storage throughput. For more information, see SAP quick sizer.

Next steps

Application design

Data platform

Networking and connectivity

Security

Operational procedures

SAP workload data platform

Article • 01/09/2023

The data platform refers to the data store technologies that support an SAP workload. SAP workloads place high demands on the data platform. We outline best practices for optimizing cost while maintaining performance.

Optimize data costs

We recommend optimizing the storage cost for your SAP workload. Storage is an essential component of an SAP workload. Storage contains active data and backup data that is critical to your organization. Storage affects the performance, availability, and recoverability of an SAP workload. It's important to have the right performance at the right cost. Here are recommendations to help you reach this goal.

Use reserved capacity storage type. There are several storage options available to choose from based on the workload requirement. Managed disks, blog storage, and backup storage can support an SAP workload in various combinations. Each of these options comes with storage reservation options that lower overall costs for persistent data.

For more information, see:

- Azure disk reserved capacity
- Blob storage reserved capacity
- Azure Backup Storage reserved capacity

Use lifecycle management policies. Other than reserved capacity, you need to ensure the data-retention period is right for the SAP workload. An SAP database backup can be large and add to the storage cost if not optimized. We recommend that you create a lifecycle policy that meets the recovery time objective (RTO) and recovery point objective (RPO) of your SAP workload. The policy should move into Premium, Standard, Cold, Archive storage based on its age and business requirements.

Improve data reliability

Data reliability is essential for ensuring continuity of operations. We provide reliability recommendations for configuring database reliability, creating SAPMNT share reliability, using backups, and implementing a disaster recovery solution.

Configure database reliability

An SAP application feeds data to multiple enterprise systems, making database resiliency a key workload consideration. We recommend replicating production data for the highest resiliency. Cross-region replication is the preferred disaster recovery solution. But for a more affordable option, you should configure zone redundancy at a minimum. The methods you choose depends on the database management system (DBMS) and required business service-level agreement (SLA). Below are recommendations for the database layer.

Define RPO and RTO. Creating database resiliency requires a plan to recover data loss. A logical error on the SAP database, a large-scale disaster, or a system outage can cause data loss in an SAP workload. Your recovery plan should identify how much data you're willing to lose and how fast you need to recover. The amount of data loss you're willing to lose is your recovery point objective (RPO). How fast you need to recover is your recovery time objective (RTO). When you design for recoverability, you need to understand the desired and actual RPO and RTO of your SAP application.

Use synchronous replication for no data loss. In some scenarios, there's no tolerance for data loss. The recovery point objective is 0. To achieve this RPO, you need use synchronous replication on the database layer. Synchronous replication commits database transactions to database instances in two separate zones or regions. You should measure the latency between the two instances to ensure it meets workload needs, and you can do it with the SAP `niping` measuring tool. Higher network latency will slow down the scalability of your workload, and physical distance between the instances adds network latency. As a result, replication across regions will have higher latency than across availability zones because there's more distance between the instances. Database replication between different regions should be asynchronous and replication between availability zones should be synchronous. It's important to balance resiliency and latency in SAP workload design.

For more information, see:

- General Azure Virtual Machines DBMS deployment for SAP workload
- High-availability architecture and scenarios for SAP NetWeaver
- Network latency between and within zones

Create SAPMNT share reliability

SAPMNT hosts the physical kernel files for SAP application and can be a single point of failure. Several options are available on Azure to created redundancy and architect a highly available SAPMNT share. We recommend using Azure Premium Files or Azure

NetApp Files for Linux and Azure Premium Files. For Windows-based deployments, you should use Azure NetApp Files or Azure Shared Disk.

There are also a few application specific configurations you should address for SAPMNT reliability. You need shared directories in the environment (`/sapmnt/SID and /usr/sap/trans`) to deploy the SAP NetWeaver application layer. We recommend creating highly available file systems and ensuring they're resilient. The `/sapmnt/SID` and `/usr/sap/SID/ASCS` directories are important. You should place these file systems on NFS on Azure Files to achieve the maximum reliability.

For more information see, NFS on Azure Files.

Table 1 - SAPMNT guidance for each operating system.

OS	SAPMNT Guidance
Windows	Cluster an SAP ASCS/SCS instance on a Windows failover cluster by using a cluster shared disk in Azure Cluster an SAP ASCS/SCS instance on a Windows failover cluster by using a file share in Azure High availability for SAP NetWeaver on Azure VMs on Windows with Azure Files Premium SMB for SAP applications High availability for SAP NetWeaver on Azure VMs on Windows with Azure NetApp Files(SMB) for SAP applications
Red Hat Enterprise Linux (RHEL)	High availability for SAP NetWeaver on Azure VMs on Red Hat Enterprise Linux with NFS on Azure Files Azure Virtual Machines high availability for SAP NetWeaver on Red Hat Enterprise Linux with Azure NetApp Files for SAP applications
SUSE Linux Enterprise Server (SLES)	High-availability SAP NetWeaver with simple mount and NFS on SLES for SAP Applications VMs High availability for SAP NetWeaver on Azure VMs on SUSE Linux Enterprise Server with NFS on Azure Files

Use data backups

The SAP workload should implement a regular backup solution. Backups are the backbone of disaster recovery and help ensure continuity of operations. We have a few recommendations for backup reliability.

Start with Azure Backup

We recommend you use Azure Backup as the foundational backup strategy for an SAP workload. Azure Backup is the native backup solution in Azure, and it provides multiple capabilities to help streamline your SAP backups. With Azure Backup, we want to point out a few features.

Table 2 - Azure Backup features

Feature	Description
Native database backup compatibility	Azure Backup provides native backups through the Backint connector for SAP HANA, SQL Server, and Oracle databases used by SAP Applications. Azure backup for SAP offers an API called Backint. Backint allows backup solutions to create backups directly on the database layer. Azure backup also supports the database backup capability for HANA & SQL Server databases today.
Storage backup	The storage backup feature can help optimize the backup strategy by using disk snapshots of Azure Premium storage for selective disks. For more information on application-consistent backups, see snapshot consistency.
Virtual machine backup	Back up and restore Azure VM data through the Azure portal. Cross-region restoration lets you restore Azure VMs that were to a paired secondary region.
Long-term retention	Azure Backup allows you to retain SAP backups years for compliance and audit needs.
Backup management	Azure Backup enables you to manage backups from the Azure portal with an easy user interface.

For more information, see:

- Azure Backup documentation
- SAP HANA backup overview
- Backup guide for SAP HANA on Azure Virtual Machines
- Backup guide for SQL Server on Azure Virtual Machines

Find marketplace backup solutions

Several certified third-party backup solutions exist in the Azure Marketplace . These solutions offer vendor backup capabilities and SAP-certified backup capabilities. You should consider layering these solutions on top of Azure Backup to generate custom solutions with foundational support.

Microsoft partners provide solutions that are integrated with Azure Storage for archive, backup, and for business continuity and disaster recovery (BCDR) workloads. The partner solutions take advantage of the scale and cost benefits of Azure Storage. You can use the solutions to help solve backup challenges, create a disaster recovery site, or archive unused content for long-term retention. They can replace tape-based backups and offer an on-demand economic recovery site with all the compliance standards and storage features such as immutable storage and lifecycle management.

Use snapshots

A snapshot is a point-in-time, copy of your data. The speed and reliability of snapshots can help manage large databases and protect the primary database against corruption or failure. These features make snapshots critical for disaster recovery. We have a few options to create and store backups for your SAP workload.

Azure Backup can take database backups for HANA and SQL Server, for example. The Backup vault feature of Azure Shared Disk can serve as your database storage solution. Azure NetApp Files (ANF) can also back up critical data by using snapshots, such as ANF volumes Snapshot. ANF Cross Region Replication uses ANF snapshots to replicate data from one region to another.

The right solution depends on your desired cost and availability levels. In some scenarios, you might want to replicate your SAP on Azure data to other Azure regions for disaster recovery. However, you can achieve the same capabilities with Azure Storage replication, such as Geo-redundant storage (GRS) or Azure Site Recovery.

For more information, see:

- SAP workload configurations with Azure Availability Zones
- SAP NetWeaver disaster recovery
- Azure Site Recovery for SAP workloads
- Azure Storage redundancy
- Back up SAP HANA databases' instance snapshots in Azure VMs

Implement a disaster recovery plan

We recommend you invest in disaster recovery (DR) to improve the reliability of the SAP workload. Disaster recovery is achieved by replicating primary data to a secondary location. Several tools & methodology can be used to the achieve goal. Disaster Recovery is required when the primary location isn't accessible due to technical or natural disaster. Disaster Recovery solutions can be across zones within region or across regions based on your business requirements, but we recommended DR across region for better resiliency.

For more information, see:

- Azure Site Recovery
- Cross-region replication of Azure NetApp Files volumes
- Cross-region snapshot copy for Azure Disk Storage
- Backup and Disaster Recovery

Improve storage performance

It’s important to choose the appropriate storage solutions to support the data needs of the SAP workload. The correct solution can improve the performance of existing capabilities and allow you to add new features. In general, storage needs to meet the input/output operations per second (IOPS) requirements and throughput needs of the SAP database.

For more information, see storage types for an SAP workload.

Use storage that supports performance requirement. Microsoft supports different storage technology to meet your performance requirement. For SAP workload, you can use Azure Managed Disk (for example, Premium SSD, Premium SSD v2, Standard SSD) and Azure NetApp Files.

Configure storage for performance. We've published a storage configuration guideline for SAP HANA databases. It covers production scenarios and a cost-conscious non-production variant. Following the recommended storage configurations will ensure the storage passes all SAP hardware and cloud measurement tool (HCMT) KPIs. For more information, see SAP HANA Azure virtual machine storage configurations.

Enable write accelerator. Write accelerator is a capability for M-Series VMs on Premium Storage with Azure Managed Disks exclusively. It’s imperative to enable write accelerator on the disks associated with the /hana/log volume. This configuration facilitates sub millisecond writes latency for 4 KB and 16-KB blocks sizes. For more information, see Azure Write Accelerator.

Choose the right VM. Choosing the right VM has cost and performance implications. The goal is to pick a storage VM that supports the IOPS and throughput requirements of the SAP workload. There are three critical areas to focus while selecting a VM

Table 3 - Compute features that affect performance

Compute features	Description
Number of vCPUs	The number of CPUs has a direct effect on the licenses in the database node. Most of the databases follow a core-based licensing model. Use the amount that meets your needs and adjust licensing agreements as necessary.
Memory	Memory is critical to application performance, and your SAP application can have high memory demands. In general, higher memory provides more memory-reads, less paging, and higher VM cost.
Throughput	Throughput is important for an application hosted on one of the VMs to communicate with outside the VM by using its network interface cards (NICs).

Next steps

Application design

Application platform

Networking and connectivity

Security

Operational procedures

SAP workload networking and connectivity

Article • 01/09/2023

Networking for an SAP workload has many facets, but we want to focus on the performance affects of networking. The goal is to help you make the right networking decisions to ensure peak SAP workload performance.

Networking performance

An SAP workload needs to communicate with other workloads. Common communication paths are to local storage, external storage, NICs, VMs in the network, VMs in other networks, and third-party applications. Optimize workload networking to improve these communication channels to meet workload and application demand. If SAP network performance isn't considered, it will cause application performance issues.

Understand proximity placement groups. Proximity placement groups reduce the distance between SAP workloads. They can group different VM types under a single network spine. As the Azure footprint grows, a single availability zone may span multiple physical data centers. The distribution across data centers can create network latency impacting your SAP application performance. The proximity of the VMs provides lower network latency.

The capabilities are ideal, but there are drawbacks to be aware of. Proximity placement groups often limit your VM choices and make resizing VMs more difficult. Proximity placement groups bind VMs to a specific network spine. This binding limits the possible combinations of different VM types. The host hardware that is needed to run a certain VM type might not be present in the data center or under the network spine to which the proximity placement group was assigned. The availability of VM types can be severely restricted.

We recommend using proximity placement groups in two scenarios. (1) Use proximity placement groups in Azure regions where latency across zones is higher than recommended for the SAP workload. (2) Use proximity placement groups for application volume group. The Application Volume Group feature of Azure NetApp Files (ANF) uses PPG to deploy ANF volumes close to the VM/compute cluster. We recommend using this feature as designed.

For more information, see:

- Overview of proximity placement groups
- SAP and proximity placement groups

Use accelerated networking. Accelerated Network is default for most of the VM deployments and is recommended for every VM hosting an SAP workload. Accelerated Network improves the network performance by bypassing the physical switch. We recommend you enable Accelerated Networking on the Azure VMs running your SAP Application and Database. Accelerated networking provides improved latency, jitter, and CPU utilization. You should test the latency between the SAP application server and database with the SAP ABAP report /SSA/CAT. It's an Inventory Check for the SAP Azure Workbook. For more information, see accelerated networking overview.

On-premises connectivity

Use ExpressRoute GlobalReach. ExpressRoute is a private and resilient way to connect your on-premises networks to different Azure regions. This feature allows you to link ExpressRoute circuits to make a private network between your on-premises networks. Global Reach should be used for SAP HANA Large Instance deployments to enable direct access from on-premises to your HANA Large Instance units deployed in different regions. For more information, see ExpressRoute Global Reach.

Next steps

Application design

Application platform

Data platform

Security

Operational procedures

SAP workload security

Article • 01/09/2023

Azure provides all the tools needed to secure your SAP workload. SAP applications can contain sensitive data about your organization. You must protect your SAP architecture with secure authentication methods, hardened networking, and encryption.

Configure identity management

Identity management is a framework to enforce the policies that control access to critical resources. Identity management controls access your SAP workload within or outside its virtual network. There are three identity management use cases to consider for your SAP workload, and the identity management solution differs for each.

Use Azure Active Directory

Organizations can improve the security of Windows and Linux virtual machines in Azure by integrating with Azure Active Directory (Azure AD). Azure AD is a fully managed identity and access management service. Azure AD can authenticate and authorize end user's access to the SAP operating system. You can use Azure AD to create domains that exist on Azure, or use it integrate with your on-premises Active Directory identities. Azure AD also integrates with Microsoft 365, Dynamics CRM Online, and many Software-as-a-Service (SaaS) applications from partners. We recommend using System for Cross-Domain Identity Management (SCIM) for identity propagation. This pattern enables optimal user life cycle.

For more information, see:

- SCIM synchronization with Azure Active Directory
- Configure SAP Cloud Platform Identity Authentication for automatic user provisioning
- Azure Active Directory Single sign-on (SSO) integration with SAP NetWeaver
- Sign in to a Linux virtual machine in Azure by using Azure AD and OpenSSH
- Sign in to a Windows virtual machine in Azure by using Azure AD

Configure single sign-on

You can access the SAP application with the SAP frontend software (SAP GUI) or a browser with HTTP/S. We recommend configuring single sign-on (SSO) using Azure

Active Directory or Active Directory Federation Services (AD FS). SSO allows end users to connect to SAP applications via browser where possible.

For more information, see:

- SAP HANA SSO
- SAP NetWeaver SSO
- SAP Fiori SSO
- SAP Cloud Platform SSO
- SuccessFactors SSO
- Azure Active Directory overview

Use application-specific guidance

We recommend consulting the SAP Identity Authentication Service for SAP Analytics Cloud, SuccessFactors, and SAP Business Technology Platform. You can also integrate services from the SAP Business Technology Platform with Microsoft Graph using Azure AD and the SAP Identity Authentication Service.

For more information, see:

- Using Azure Active Directory to secure access to SAP platforms and applications.
- SAP Identity Authentication Service
- SAP Identity Provisioning Service

A common customer scenario is deploying SAP application into Microsoft Teams. This solution requires SSO with Azure AD. We recommend browsing the Microsoft commercial marketplace to see which SAP apps are available in Microsoft Teams. For more information, see the Microsoft commercial marketplace.

Table 1 - Summary of the recommended SSO methods

SAP solution	SSO method
SAP NetWeaver based-web applications such as Fiori, WebGui	Security Assertion Markup Language (SAML)
SAP GUI	Kerberos with windows active directory or AAD-Domain services or third party solution
SAP PaaS and SaaS applications such as SAP Business Technology Platform (BTP), Analytics Cloud, Cloud Identity Services, SuccessFactors, Cloud for Customer, Ariba	SAML / OAuth / JSON Web Tokens (JWT) and pre-configured authentication flows with Azure AD directly or by proxy with the SAP Identity Authentication Service

Use role-based access control (RBAC)

It's important to control access to the SAP workload resources you deploy. Every Azure subscription has a trust relationship with an Azure AD tenant. We recommend you use Azure role-based access control (Azure RBAC) to grant users within your organization access the SAP application. Grant access by assigning Azure roles to users or groups at a certain scope. The scope can be a subscription, a resource group, or a single resource. The scope depends on the user and how you've grouped your SAP workload resources.

For more information, see:

- Azure AD trust relationship
- Azure RBAC

Enforce network and application security

Network and application security controls are baseline security measures for every SAP workload. Their importance bears repeating to enforce the idea that the SAP network and application requires rigorous security review and baseline controls.

Use hub-spoke architecture. It's critical to differentiate between shared services and SAP application services. A hub-spoke architecture is a good approach to security. You should keep workload specific resources in its own virtual network separate from the shared services in hub such as management services and DNS.

For SAP-native setups, you should use SAP Cloud Connector and SAP Private Link for Azure as part of the hub-spoke setup. These technologies support the SAP extension and innovation architecture for the SAP Business Technology Platform (BTP). Azure native integrations fully integrated with Azure virtual networks and APIs and don't require these components.

Use network security groups. Network security groups (NSGs) allow you to filter network traffic to and from your SAP workload. You can define NSG rules to allow or deny access to your SAP application. You can allow access to the SAP application ports from on-premises IP addresses ranges and denying public internet access. For more information, see network security groups

Use application security groups. In general, the security best practices for application development also apply in the cloud. These include things like protecting against cross-site request forgery, thwarting cross-site scripting (XSS) attacks, and preventing SQL injection attacks.

Application security groups (ASGs) make it easier to configure the network security of a workload. The ASG can be used in security rules instead of explicit IPs for VMs. The VMs are then assigned to ASG. This configuration supports the reuse of the same policy over different application landscapes, because of this abstraction layer. Cloud applications often use managed services that have access keys. Never check access keys into source control. Instead, store application secrets in Azure Key Vault. For more information, see application security groups.

Filter web traffic. An internet facing workload must be protected using services like Azure Firewall, Web Application Firewall, Application Gateway to create separation between endpoints. For more information, see inbound and outbound internet connections for SAP on Azure.

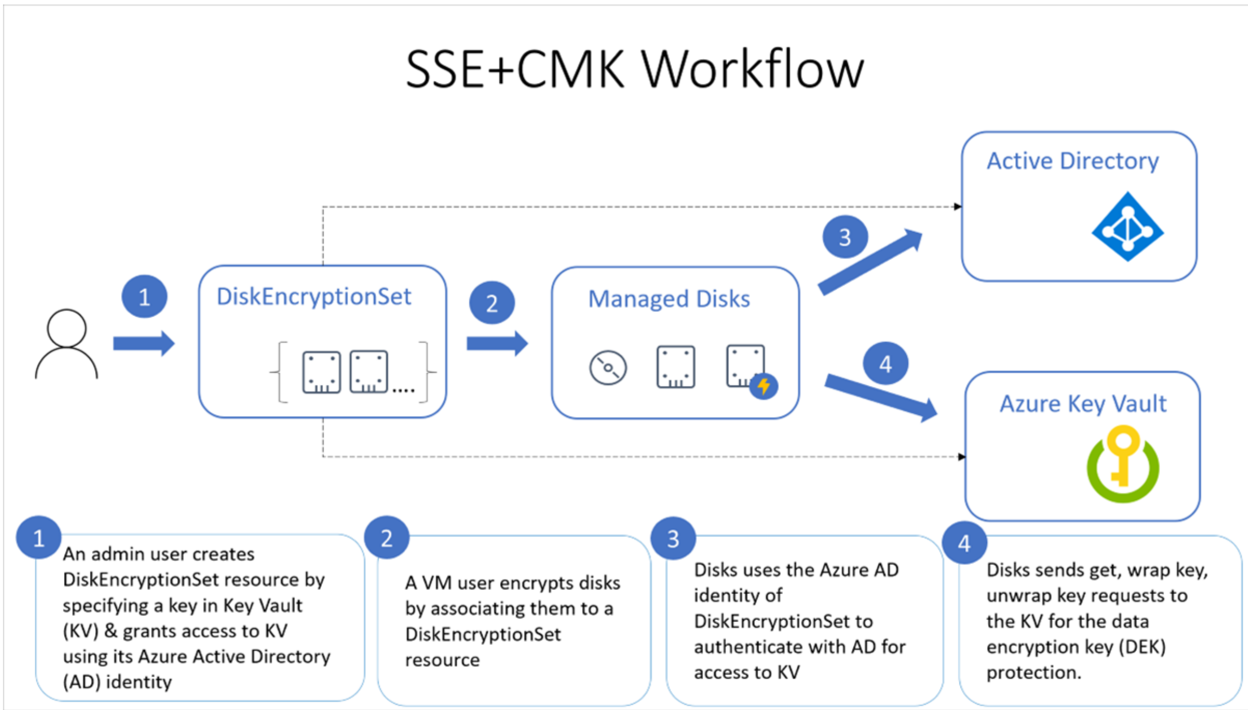
Encrypt data

Azure includes tools to safeguard data according to your organization's security and compliance needs. It's essential that you encrypt SAP workload data at rest and in transit.

Encrypt data at rest

Encrypting data at rest is a common security requirement. Azure Storage service-side encryption is enabled by default for all managed disks, snapshots, and images. Service-side encryption uses service-managed keys by default, and these keys are transparent to the application.

We recommend you review and understand service/server-side encryption (SSE) with customer-managed keys (CMKs). The combination of server-side encryption and a customer-managed key allows you to encrypt data at rest in the operating system (OS) and data disks for available SAP OS combinations. Azure Disk Encryption doesn’t support all SAP operating systems. The customer-managed key should be stored in Key Vault to help ensure the integrity of the operating system. We also recommend encrypting your SAP databases. Azure Key Vault supports database encryption for SQL Server from the database management system (DBMS) and other storage needs. The following image shows the encryption process.



When you use client-side encryption, you encrypt the data and upload the data as an encrypted blob. Key management is done by the customer. For more information, see:

- Server-side encryption for managed disks
- Azure Storage service-side encryption
- Service-side encryption using customer-managed key in Azure Key Vault
- Client-side encryption

Encrypt data in transit

Encryption in transit applies to the state of data moving from one location to another. Data in transit can be encrypted in several ways, depending on the nature of the connection. For more information, see encryption of data in transit.

Collect and analyze SAP application logs

Application log monitoring is essential for detecting security threats at the application level. We recommend using the Microsoft Sentinel Solution for SAP. It's a cloud-native security information and event management (SIEM) solution built for your SAP workload running on a VM. For more information, see Microsoft Sentinel Solution for SAP.

For general security information, see:

- Azure security documentation
- Trusted Cloud eBook

Next steps

Application design

Application platform

Data platform

Networking and connectivity

Operational procedures

SAP workload operational procedures

Article • 01/09/2023

Standard operating procedures (SOPs) are documented processes for managing a workload. Each SAP workload should have SOPs to govern operations. Without SOPs, teams drift from management best practices, so we recommend a continuous cycle of assessment and health checks for your SAP workload.

Use health checks and assessments

Run health checks. We have four Azure SAP (AzSAP) health checks: (1) deployment checklist, (2) inventory checklist, (3) quality checks, and (4) Linux VM OS analyzer. The image below shows how they share a cycle with our Azure SAP assessments. For more information on the health checks, see SAP quality checks.

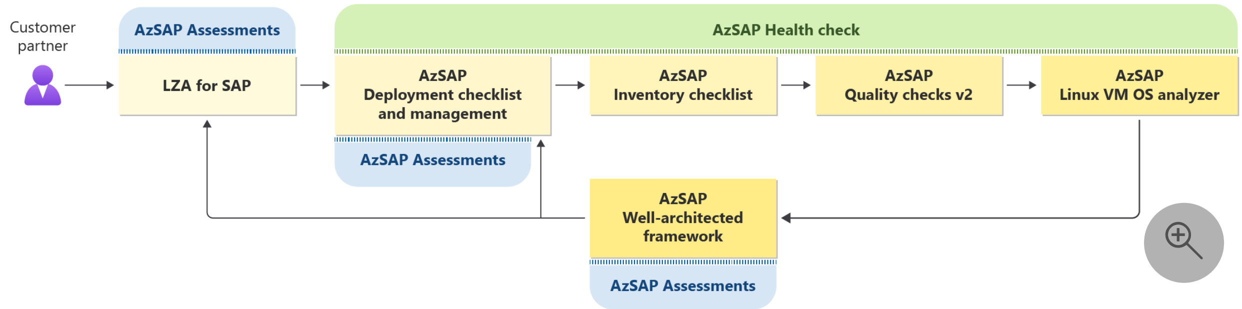


Figure 1: The cycle of SAP health checks and assessments throughout th journey.

Conduct assessments. We have three SAP assessments: (1) landing zone accelerator (LZA), (2) Azure SAP (AzSAP) deployment management assessment, and (3) the AzSAP Well-architected framework assessment. These assessments are designed for different stages in the SAP workload lifecycle.

The AzSAP Well-architected framework assessment is for operations. It compares your SAP operations against SAP workload best practices. The assessment encourages continuous improvement by building on each previous assessment.

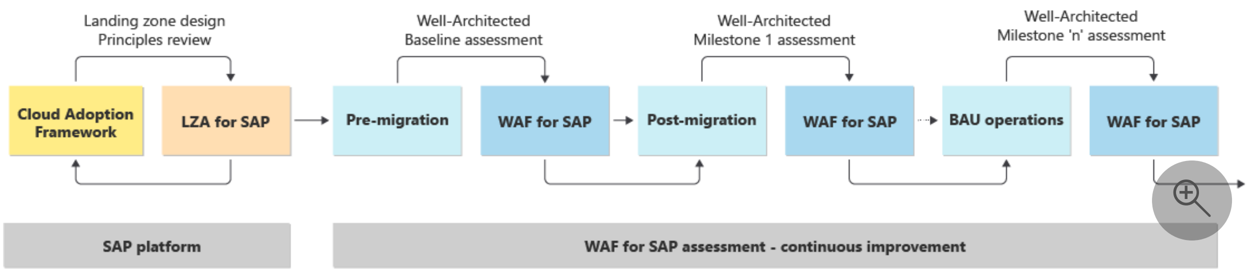


Figure 2: How the Well-architected assessment creates milestones and builds on these milestones over time.

The initial assessment creates a baseline, and the next iteration of assessment uses the previous assessment as the starting point. It will maintain the selections from the last assessment to track and review the design principle. Because the assessment builds on itself, you can track improvements overtime. The assessment is designed for an existing SAP workload in Azure and can assess one or more of the WAF pillars.

We recommend using this SAP assessment to develop and realign the SOPs for your SAP workload. The assessment identifies areas of strength and weakness that allow you to build better SOPs. For more information, see Azure Well-Architected Review.

Monitor the workload

Monitoring is the process of collecting, analyzing, and acting on data gathered from an SAP workload. Monitoring provides insights of the health of the workload to compare with an expected baseline. It allows you to know when, where, and why failures occur. A monitoring best practice is to use a common and consistent logging schema that lets you correlate events across systems. The monitoring and diagnostics process has several distinct phases.

Table 1 - Phases of monitoring and diagnostic process

Phases	Process description
Instrumentation	Generating the raw data from application logs, web server logs, the diagnostics built into the Azure platform, and other sources.
Collection and storage	Consolidating the data into one place.
Analysis and diagnosis	Troubleshooting issues and seeing the overall health.
Visualization and alerts	Using data to spot trends or alert your operations team.

We recommend using Azure Monitor for SAP solutions to drive these processes. Azure Monitor for SAP is an Azure-native monitoring product for SAP landscapes that run on Azure. Azure Monitor for SAP solutions uses specific parts of the Azure Monitor infrastructure to provide insights into the monitoring of SAP Netweaver, SAP HANA, SQL Server & Pacemaker High-Availability deployments on Azure. For more information, see Azure Monitor for SAP Solutions.

Automate workload infrastructure

You should use infrastructure as code (IaC) to automate SAP workload deployments with minimal human intervention and build a scalable and consistent SAP workload on Azure. The manual process of creating the required SAP workload resources is slow and allows for errors. Microsoft has a repository of SAP deployment templates that you should use. It's called the SAP on Azure Deployment Automate Framework. The templates support SAP HANA and NetWeaver with any database on any SAP-supported operating systems.

For more information, see:

- SAP deployment automation framework
- SAP automate repository
- Azure Monitor for SAP solutions

Table 2 - Benefits of automated deployments with IaC

Benefit domain	**Automate deployment benefits**	**Manual deployment disadvantages**
Knowledge	Works immediately after some initial preparation time. Requires little domain knowledge.	Requires specialized knowledge in many domains outside of SAP.
Time	Consumes predictable time from 30 minutes to a couple of hours.	Can take much more time depending on the size of the SAP landscape, depending on the size of the SAP landscape.
Cost	Makes automated deployments cheap due to less time spent.	Expensive due to more time spent.
Testing	Provides templates that include test instrumentation during deployment and migration.	Allows for limited testing. Requires more work to inject tests in the process.
Scaling	Allows you to easily scale up, down, and out. Provides new deployment templates.	Takes more time to scale and customize the environment.
Standardization	Applies your defined standards with each deployment.	Sometimes leads to unwanted variations in design.

Next steps

Application design

Application platform

Data platform

Networking and connectivity

Security

Sustainability workload documentation

In partnership with the Green Software Foundation, we've developed this set of recommendations for optimizing Azure workloads. This documentation helps you plan your path forward, improve your sustainability posture, and create new business value while reducing your operational footprint.

Get started

OVERVIEW

What is sustainability?

Design methodology

Design principles

Design areas

CONCEPT

Application design

Application platform

Testing

Operational procedures

Networking and connectivity

Storage

Security

Reference examples

ARCHITECTURE

Example scenario - Measure Azure app sustainability by using the SCI score

Learn

TRAINING

The Principles of Sustainable Software Engineering

Assessment

HOW-TO GUIDE

Sustainability assessment tool

Sustainable workloads

Article • 10/12/2022

This section of the Microsoft Azure Well-Architected Framework aims to address the challenges of building sustainable workloads on Azure. Review the provided guidance that applies Well-Architected best practices as a technical foundation for building and operating sustainable solutions on Azure.

We encourage you to also read more about the Microsoft Cloud for Sustainability for opportunities to leverage the capabilities of that platform as part of your solution architecture. However, guidance found in this article series is focused on *any* solutions you're building or operating on Azure.

Additionally, read about The Carbon Benefits of Cloud Computing: a Study of the Microsoft Cloud to learn more about how Azure is more energy efficient and carbon efficient than on-premises solutions.

What is a sustainable workload?

The term *workload* refers to a collection of application resources that support a common business goal or the execution of a common business process, with multiple services, such as APIs and data stores, working together to deliver specific end-to-end functionality.

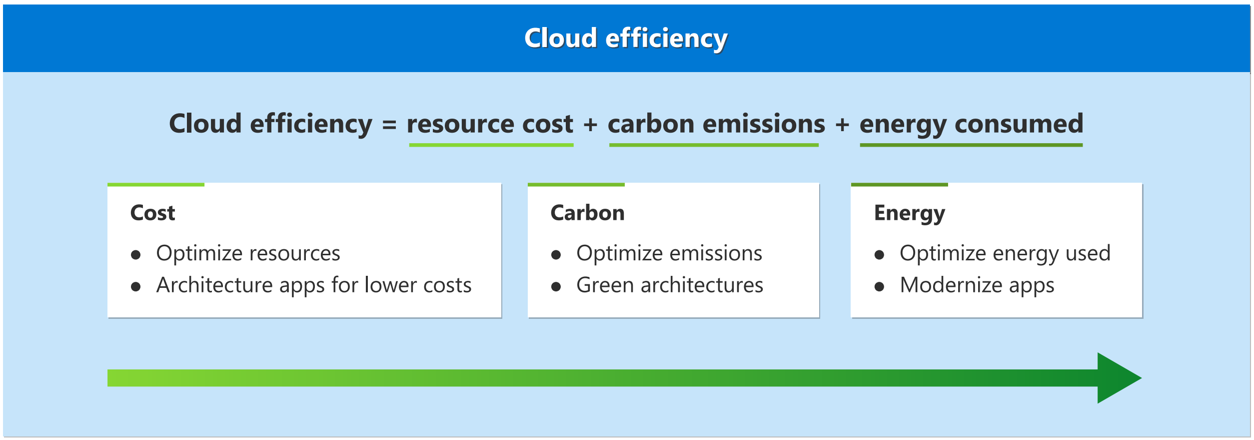
With *sustainability*, we refer to the environmental impact of our workloads.

A *sustainable workload* therefore describes the practice of designing solutions that maximize utilization while minimizing waste, ultimately reducing the footprint on the environment.

Cloud efficiency overview

Making workloads more cloud efficient requires combining efforts around cost optimization, reducing carbon emissions, and optimizing energy consumption. Optimizing the application's cost is the initial step in making workloads more sustainable.

Here's a conceptual overview of cloud efficiency in this context:



Scoring and measuring the cloud efficiency is essential to understand whether changes tracked over time have any impact.

Learn about building more sustainable and efficient workloads by starting with the design area for sustainable Application Design.

What are the common challenges?

Building and designing sustainable workloads on Microsoft Azure can be challenging for these main reasons:

- Understanding if your workloads are in alignment with sustainability targets. It requires assessments of current workloads to determine if they're designed in a sustainable way.
- Designing workloads that are natively environmentally friendly and optimized.
- Measuring and tracking the emissions of your workloads.

Is sustainability only about performance and cost?

While performance efficiency and cost optimization are areas of strong focus for designing sustainable workloads, the other pillars of the Well-Architected Framework are equally important when building long-term sustainable workloads on Azure.

- Security: how the security appliances in a workload are optimized and designed to auto-scale will have an impact on the environment.
- Reliability: designing reliable workloads that meet sustainability guidelines from the Green Software Foundation can greatly reduce the workloads' carbon and electricity footprint.
- Operational Excellence: how a workload is able to effectively respond to operational issues can ultimately reduce carbon emissions.

What are the key design areas?

Sustainable guidance within this series is composed of architectural considerations and recommendations oriented around these key design areas.

Decisions made in one design area can impact or influence decisions across the entire design. The focus is ultimately on building a sustainable solution to minimize the footprint and impact on the environment.

Design area	Description
Application design	Cloud application patterns that allow for designing sustainable workloads.
Application platform	Choices around hosting environment, dependencies, frameworks, and libraries.
Testing	Strategies for CI/CD pipelines and automation, and how to deliver more sustainable software testing.
Operational procedures	Processes related to sustainable operations.
Storage	Design choices for making the data storage options more sustainable.
Network and connectivity	Networking considerations that can help reduce traffic and amount of data transmitted to and from the application.
Security	Relevant recommendations to design more efficient security solutions on Azure.

We recommend that readers familiarize themselves with these design areas, reviewing provided considerations and recommendations to better understand the consequences of encompassed decisions.

Next step

Review the sustainability design methodology.

Design methodology

Design methodology for sustainable workloads on Azure

Article • 10/21/2022

Building a sustainable application on any cloud platform requires technical expertise and an understanding of sustainability guidelines in general and for your specific cloud platform.

This design methodology aims to help establish an understanding about producing more carbon efficient solutions, measuring your carbon impact, and ultimately reducing unnecessary energy usage and emissions.

1—Design for business requirements

Businesses globally have different requirements. Expect that the review considerations and design recommendations provided by this design methodology will yield different design decisions and trade-offs for different scenarios and organizations.

Establish your business requirements and priorities, then review the design methodologies in alignment with those requirements.

2—Evaluate the design areas using the design principles

Refer to the sustainability design principles and the design areas below for your sustainability workloads.

Decisions made within each design area will echo across other design areas. Review the considerations and recommendations in each design area to understand the consequences and impact and any known trade-offs.

Design areas:

- Application design
- Application platform
- Deployment and testing
- Operational procedures
- Storage
- Network and connectivity
- Security

3—Understanding your emissions

To lower your emissions, you need to understand how to measure your sustainability efforts.

Briefly about emission scopes

At Microsoft, we segment our greenhouse gas (GHG) emissions into three categories, consistent with the Greenhouse Gas Protocol.

- **Scope 1 emissions**: direct emissions that your activities create.
- **Scope 2 emissions**: indirect emissions that result from the production of the electricity or heat you use.
- **Scope 3 emissions**: indirect emissions from all other activities you're engaged in. For a business, these Scope 3 emissions can be extensive. They must be accounted for across its supply chain, materials in its buildings, employee business travel, and the life cycle of its products (including the electricity customers consume when using the products). A company's Scope 3 emissions are often far more significant than its Scope 1 and 2 emissions combined.

As a customer, the context of Scope 3 emissions can be network configuration and delivery, power consumption, and devices outside the data center. If an application uses excess bandwidth or packet size, it will impact from when the traffic leaves the data center, through the various hops on the internet, down to the end-user device. Reducing network bandwidth, therefore, can have a significant impact throughout the delivery chain. The same considerations apply to compute resources, data storage, application platform decisions, application design, and more.

Find more in-depth details and definitions in Azure's Scope 3 Methodology White Paper, published in 2021.

Measure and track carbon impact

Microsoft aligns with the Green Software Foundation, responsible for creating the Software Carbon Intensity (SCI) specification.

To measure the carbon impact of an application, the GSF provided a scoring methodology called SCI, calculated as follows:

`SCI = ((E*I)+M) per R`

Where:

- `E` = Energy consumed by a software system. Measured in kWh.
- `I` = Location-based marginal carbon emissions. Carbon emitted per kWh of energy, gCO2/kWh.
- `M` = Embodied emissions of a software system. Carbon that is emitted through the hardware on which the software is running.
- `R` = Functional unit, which is how the application scales; per extra user, per API call, per service, etc.

With this knowledge, it's essential to consider not only the application infrastructure and hardware but also the user devices and application scalability, as it can alter the environmental footprint considerably.

Read the full SCI specification on GitHub.

Carbon tracking and reporting with the Emissions Impact Dashboard

Microsoft offers the Emissions Impact Dashboard for Azure and Microsoft 365, which helps you measure your cloud-based emissions and carbon savings potential.

We recommend you use this tool to get the insights and transparency you need to understand your carbon footprint and to measure and track emissions over time.

Download the Emissions Impact Dashboard Power BI app for Azure to get started.

Leverage the Microsoft Sustainability Manager

Customers using Microsoft Cloud for Sustainability can leverage Microsoft Sustainability Manager. This extensible solution unifies data intelligence and provides comprehensive, integrated, and automated sustainability management for organizations at any stage of their sustainability journey. It automates manual processes, enabling organizations to record, report, and reduce their emissions more efficiently.

Use a proxy solution to measure emissions

One way of estimating the carbon emissions from workloads is to design a proxy solution architecture based on the SCI model as described above.

Defining the proxies for applications can be done in different ways. For example, using these variables:

- Any known carbon emission of the infrastructure

- The cost of the infrastructure
- Edge services and infrastructure carbon emissions
- The number of users that are concurrently using the application
- Metrics of the application to inform us about the performance over time

By designing an equation using the above variables, you can estimate the carbon score (an approximation), helping you understand if you're building sustainable solutions.

There's also the aspect of application performance. You can link performance to cost and carbon and assume this relationship yields a value. With this relation, you can simplify the view like this:

Application performance	Application cost	Likely outcome
High	Unchanged	Optimized app
High	Lower	Optimized app
Unchanged/Lower	Higher	According to the green principles, a higher energy cost can cause higher carbon emissions. Therefore, you can assume that the app produces unnecessary carbon emissions.
High	High	The app may be producing unnecessary carbon

Therefore, building a carbon score dashboard can make use of the following proxies:

- Cost
- Performance
- Carbon emissions of the infrastructure (if known/available)
- Usage over time (requests, users, API calls, etc.)
- Any extra measurement that is relevant to the application

4—The shared responsibility model for sustainability

Reducing emissions is a shared responsibility between the cloud provider and the customer designing and deploying applications on the platform.

Ways to reduce emissions

Reducing carbon emissions can happen with three possible solutions:

- Carbon neutralization; compensating carbon emissions

- Carbon avoidance; not emitting carbon in the first place
- Carbon removal; subtract carbon from the atmosphere

The goal of green software is to avoid unnecessary emissions in the first place, hence actively working toward a more sustainable future. Further, *carbon removal* is the preferred goal for removing emissions from our atmosphere.

Microsoft is committed to being carbon negative by 2030, and by 2050 to have removed all the carbon the company has emitted since it was founded in 1975.

A shared responsibility

As a cloud provider, Microsoft is responsible for the data centers hosting your applications.

However, deploying an application in the Microsoft cloud doesn't automatically make it sustainable, even if the data centers are optimized for sustainability. Applications that aren't optimized may still emit more carbon than necessary.

Let's take an example.

You deploy an app to an Azure service, but you only utilize 10% of the allocated resources. The provisioned resources are underutilized, ultimately leading to unnecessary emissions.

It would help if you considered scaling to an appropriate tier of the resource (rightsizing) or deploying more apps to the same provisioned resources.

We recommend making applications more efficient to utilize the data center capacity in the best way possible. Sustainability is a shared responsibility goal that must combine the efforts of the cloud provider and the customers in designing and implementing applications.

Next steps

Review the design principles for sustainability.

Design principles

Design principles of a sustainable workload

Article • 10/12/2022

The sustainability design methodology provides a framework to record, report, and reduce or optimize the environmental impact of your workloads.

To achieve an increase in carbon efficiency, consider how your workload, directly and indirectly, can reduce carbon emissions through:

- Using less physical and virtual resources
- Using less energy
- Using energy and resources more intelligently
- Supporting older devices

It's important to effectively record, report, and reduce carbon emissions through actionable insights.

- Gain transparency into your current carbon impact
- Estimate savings
- Take action to accelerate progress

These critical design principles for sustainability resonate and extend the quality pillars of the Azure Well-Architected Framework—Reliability, Security, Cost Optimization, Operational Excellence, and Performance Efficiency.

Principles of green software

Microsoft is actively working toward sustainability targets, and empowers every organization to help reduce emissions and improve our environmental health. The Azure Well-Architected Framework workload for sustainability aligns with the Green Software Principles from the Green Software Foundation.

The principles of green software are the starting point to understand the SCI model and how this will be included in our framework.

Carbon efficiency

Principle: Emit the least amount of carbon possible.

The application or software must emit the least amount of carbon possible. A carbon efficient cloud application is one that is optimized, and the starting point is the cost – streamlining the application infrastructure and cost will ensure that no unnecessary resources are wasted in the cloud to run the software. But this isn't enough, as you might have cost-optimized your application but still waste tons of resources that emit carbon for no reason.

- Read more about the Carbon Efficiency principle from the Green Software Foundation.

Energy efficiency

Principle: Use the least amount of energy possible.

The goal of this principle is that you build applications that are energy-efficient. This is a common pattern for mobile applications, since they must rely on a battery powered device and are optimizing its consumption. It's less common, however, for desktop or web applications, since until now, developers have never been asked to optimize the electricity consumption of their software.

- Read more about the Energy Efficiency principle from the Green Software Foundation

Carbon awareness

Principle: Do more when the electricity is cleaner and do less when the electricity is dirtier.

We need to make the application aware of how much carbon it's emitting. This way, we can react to specific conditions of energy supply using demand shifting and demand shaping techniques:

Technique	Description
Demand shifting	Demand shifting means moving the workloads and resources to regions or data centers, or a time in the data center where the energy supply is high and the demand is lower and can be met by renewable energy. Delaying running apps to a time when there's less demand should result in lower carbon intensity.
Demand shaping	Demand shaping means changing the application's behavior and appearance to match the energy supply in real-time. A good practice is to build an eco-version of the app and keep it as a benchmark for demand shaping and carbon optimization.

- Read more about the Carbon Awareness principle from the Green Software Foundation.

Hardware efficiency

Principle: Use the least amount of embodied carbon possible.

Embodied carbon is the carbon that was emitted to build a device. Therefore, a sustainable application will make sure older devices are supported and will maximize the efficiency of each device. The goal is to build hardware-efficient applications.

Consider the tradeoff that older devices can have power inefficiencies, and may not always be suitable.

- Read more about the Hardware Efficiency principle from the Green Software Foundation.

Measuring sustainability

Principle: What you can't measure, you can't improve.

Measuring carbon emissions of a cloud application is a complex task, as it involves the whole ecosystem of the software: from the cloud infrastructure (where we have the emissions dashboards to help us out), to the network path that is crossed, to the edge technology and user devices. With the SCI, we aren't targeting a discrete measurement of carbon emissions, but a score that will change over time and with our optimization techniques.

- Read more about the Measurement from the Green Software Foundation.

Climate commitments

Principle: Understand the exact mechanism of reduction.

Many corporations and groups have made commitments to the climate. They actively work toward new sustainability goals with a primary objective to remove, reduce and prevent carbon emissions.

There are several options for reducing the carbon footprint of any organization or entity. However, and aligned with the goal of the Green Software Foundation, our main direction should always be to avoid emitting carbon in the first place. This is what we call Abatement, or Carbon Elimination.

Once we've pursued this goal, there will still be emissions that can't be avoided. All the remaining carbon reduction methodologies will help us do so, offsetting (either compensating or neutralizing carbon).

Your company's strategy can be a mix of all the possible methodologies and, depending on the final result, can reach a Net Zero target when carbon emissions are eliminated where possible and the residual emissions compensated.

The SCI equation aims to eliminate emissions, which should always be the primary goal of a sustainable workload, and the score can only be reduced with abatement.

- Read more about the Climate Commitments from the Green Software Foundation.

Next steps

Review the considerations for application design.

Application design

Application design of sustainable workloads on Azure

Article • 10/12/2022

When building new or updating existing applications, it's crucial to consider how the solution will impact the climate and if there are ways to improve and optimize. Learn about considerations and recommendations to optimize your code and applications for a more sustainable application design.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Code efficiency

Demands on applications can vary, and it's essential to consider ways to stabilize the utilization to prevent over- or underutilization of resources, which can lead to unnecessary energy spills.

Evaluate moving monoliths to a microservice architecture

Monolithic applications usually scale as a unit, leaving little room to scale only the individual components that may need it.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- Evaluate the microservice architecture guidance.
- A microservice architecture allows for scaling of only the necessary components during peak load; ensuring idle components are scaled down or in. Additionally, it may reduce the overhead and resources required for deploying monolithic applications.
- Consider this tradeoff: While reducing the compute resources required, you may increase the amount of traffic on the network, and the complexity of the application may increase significantly.

- Consider this other tradeoff: Moving to microservices can result in extra deployment overhead with numerous similarities in deployment pipelines. Carefully consider the required deployment resources for monolithic versus microservice architectures.
- Additionally, read about containerizing monolithic applications.

Improve API efficiency

Many modern cloud applications are designed to transact many messages between services and components asynchronously. Consider the format used to encode the payload data. How much information does your application need to communicate, and is there room to reduce the chattiness?

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Learn about the chatty I/O antipattern to better understand how a large number of requests can impact performance and responsiveness.
- Improve the reliability and reduce unnecessary load to your systems. Implement advanced request throttling with API Management.
- Minimize the amount of data the application returns from requests by being selective and encoding the messages. See message encoding considerations.
- Cache responses to avoid reprocessing the same type of information from the backend system unless necessary. See caching in Azure API Management.

Ensure backward software compatibility to ensure it works on legacy hardware

Consider how applications render information. Does the application need to critically serve everything in the highest quality, resulting in higher bandwidth and processing? Is there room for reducing the quality of components in the UI to serve sustainability goals better?

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Support more end-user consumer devices, like older browsers and operating systems. This backward compatibility improves hardware efficiency by reusing existing hardware instead of requiring a hardware upgrade for the solution to work.

- Consider this tradeoff: If the most recent software updates have significant performance improvements, using older software versions may not be more efficient.

Leverage cloud native design patterns

Learning about cloud-native design patterns is helpful for building applications, whether they're hosted on Azure or running elsewhere. Optimizing the performance and cost of your cloud application will also reduce its resource utilization, hence its carbon emissions.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- Leverage cloud-native design patterns when writing or updating applications.

Consider using circuit breaker patterns

Consider evaluating and preventing applications from performing operations that are likely to fail. Repeated failures can lead to overhead and unnecessary processing that you can avoid with proper design patterns.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- A circuit breaker can act as a proxy for operations that might fail and should monitor the number of recent failures that have occurred and use that information to decide whether to proceed.
- Study the Circuit Breaker pattern, and then consider how you can implement the Circuit Breaker patterns to your applications.
- Consider using Azure Monitor to monitor failures and set up alerts.

Optimize code for efficient resource usage

Applications deployed using inefficient code may result in an inherent impact on sustainability.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- Reduce CPU cycles and the number of resources you need for your application.

- Use optimized and efficient algorithms and design patterns.
- Consider the Don't repeat yourself (DRY) principle.

Optimize for async access patterns

Demands on applications can vary, and it's essential to consider ways to stabilize the utilization to prevent over- or underutilization of resources, which can lead to unnecessary energy spills.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Queue and buffer requests that don't require immediate processing, then process in batch. Designing your applications in this way helps achieve a stable utilization and helps flatten consumption to avoid spiky requests.
- Read about optimizing for async access patterns.

Evaluate server-side vs. client-side rendering

Determine whether to render on the server-side or client-side when building applications with a UI.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- Consider these benefits of server-side rendering:
  - When the server's power comes from less-polluting alternatives than the client's locale.
  - When the hardware on the server has better processing-energy ratios.
  - Can use centralized caching to reduce multiple unnecessary renders.
  - Reducing the number of browser-to-server round-trips can be particularly important when the client's device has a lossy link.
  - When the client devices are older and have slower CPUs. Users don't need to upgrade their devices to support a modern browser.

- Consider these benefits of client-side rendering:
  - When the end-user devices are more suitable, pushing the responsibility of rendering to the clients.
  - It's more efficient only to render what's needed and as requested, as opposed to rendering everything at least once.
  - There's no need for a server, as you can rely on static storage.

- Browser caching is used on the clients.

Be aware of UX design for sustainability

Consider how the UX design of a workload impacts sustainability and determine what options exist for improving energy efficiency and reducing unnecessary network load, data processing, and compute resources.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Consider reducing the number of components to load and render on pages.
- Determine whether the application can render lower-resolution images and videos.
  - Don't render full-size images as thumbnails where the browser is doing the resizing.
  - Using full-size images as thumbnails or resized images will transfer more data, unnecessary network traffic, and additional client-side CPU usage due to image resizing and pre-rendering.
- Ensuring there are no unused pages will help minimize the UX design.
- Consider search and findability. Making it easier for users to find what they're looking for helps lower the amount of data stored and retrieved.
- Consider providing a lighter UI, using fewer resources and with a lower impact on sustainability, and provide users with an informed choice.
- Save energy by offering your apps and websites in dark mode, with dark backgrounds.
- Opt for using system fonts when possible to avoid forcing clients to download additional fonts, which causes more network load.

Update legacy code

Consider upgrading or deprecating legacy code if it's not running on modern cloud infrastructure or with the latest updates.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Identify inefficient legacy code suited for modernization.
- Review if there are options to move to serverless or any of the optimized PaaS options.
- Consider this tradeoff: Updating old code that might end up being deprecated can consume valuable time.

Next step

Review the design considerations for the application platform.

Application platform

Application platform considerations for sustainable workloads on Azure

Article • 02/21/2023

Designing and building sustainable workloads requires understanding the platform where you're deploying the applications. Review the considerations and recommendations in this section to know how to make better informed platform-related decisions around sustainability.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Platform and service updates

Keep platform and services up to date to leverage the latest performance improvements and energy optimizations.

Review platform and service updates regularly

Platform updates enable you to use the latest functionality and features to help increase efficiency. Running on outdated software can result in running a suboptimal workload with unnecessary performance issues. New software tends to be more efficient in general.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Upgrade to newer and more efficient services as they become available.
- Consider backward compatibility and hardware reusability. An upgrade may not be the most efficient solution if the hardware or the OS isn't supported.
- Make use of Azure Automation Update Management to ensure software updates are deployed to Azure VMs.

Regional differences

The Microsoft Azure data centers are geographically spread across the planet and powered using different energy sources. Making decisions around where to deploy your workloads can significantly impact the emissions your solutions produce.

Learn more about sustainability from the data center to the cloud with Azure. See region-specific sustainability information in the Microsoft data center sustainability fact sheets.

Deploy to low-carbon regions

Learn about what Azure regions have a lower carbon footprint than others to make better-informed decisions about where and how our workloads process data.

Green Software Foundation alignment: Carbon efficiency

Recommendation:

- Use less carbon because the data centers where you deploy the workload are more likely to be powered by renewable and low-carbon energy sources.
- Consider these potential tradeoffs:
  - The effort and time it takes to move to a low-carbon region.
  - Migrating data between data centers may not be carbon efficient.
  - Consider the cost for new regions, including low-carbon regions, which may be more expensive.
  - If the workloads are latency sensitive, moving to a lower carbon region may not be an option.

Process when the carbon intensity is low

Some regions on the planet are more carbon intense than others. Therefore it's essential to consider where we deploy our workloads and combine this with other business requirements.

Green Software Foundation alignment: Carbon efficiency, Carbon awareness

Recommendation:

- Where you have the data available, consider optimizing workloads when knowing that the energy mix comes mostly from renewable energy sources.
- If your application(s) allow it, consider moving workloads dynamically when the energy conditions change.
  - For example, running specific workloads at night may be more beneficial when renewable sources are at their peak.

Choose data centers close to the customer

Deploying cloud workloads to data centers is easy. However, consider the distance from a data center to the customer. Network traversal increases if the data center is a greater distance from the consumer.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Consider deploying to data centers close to the consumer.

Run batch workloads during low-carbon intensity periods

Proactively designing batch processing of workloads can help with scheduling intensive work during low-carbon periods.

Green Software Foundation alignment: Carbon awareness

Recommendation:

- Where you have the data available to you, plan your deployments to maximize compute utilization for running batch workloads during low-carbon intensity periods.
- Potential tradeoffs may include the effort and time it takes to move to a low-carbon region. Additionally, migrating data between data centers may not be carbon efficient, and the cost for new regions-including low—carbon regions—may be more expensive.

Modernization

Consider these platform design decisions when choosing how to operate workloads. Leveraging managed services and highly optimized platforms in Azure helps build cloud-native applications that inherently contribute to a better sustainability posture.

Containerize workloads where applicable

Consider options for containerizing workloads to reduce unnecessary resource allocation and to utilize the deployed resources better.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Deploying apps as containers allows for bin packing and getting more out of a VM, ultimately reducing the need for duplication of libraries on the host OS.
- Removes the overhead of managing an entire VM, and allows deploying more apps per physical machine. Containerization also optimizes server utilization rates and improves service reliability, lowering operational costs. Fewer servers are needed, and the existing servers can be better utilized.
- Consider these tradeoffs: The benefit of containerization will only realize if the utilization is high. Additionally, provisioning an orchestrator such as Azure Kubernetes Services (AKS) or Azure Red Had OpenShift (ARO) for only a few containers would likely lead to higher emissions overall.

Evaluate moving to PaaS and serverless workloads

Managed services are highly optimized and operate on more efficient hardware than other options, contributing to a lower carbon impact.

Green Software Foundation alignment: Hardware efficiency, Energy efficiency

Recommendation:

- Build a cloud-native app without managing the infrastructure, using a fully managed and inherently optimized platform. The platform handles scaling, availability, and performance, ultimately optimizing the hardware efficiency.
- Review design principles for Platform as a Service (PaaS) workloads.

Use Spot VMs where possible

Think about the unused capacity in Azure data centers. Utilizing the otherwise wasted capacity—at significantly reduced prices—the workload contributes to a more sustainable platform design.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- By utilizing Spot VMs, you take advantage of unused capacity in Azure data centers while getting a significant discount on the VM.
- Consider the tradeoff: When Azure needs the capacity back, the VMs get evicted. Learn more about the Spot VM eviction policy.

Right sizing

Ensuring workloads use all the allocated resources helps deliver a more sustainable workload. Oversized services are a common cause of more carbon emissions.

Turn off workloads outside of business hours

Operating idle workloads wastes energy and contributes to added carbon emissions.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- Dev and testing workloads should be turned off or downsized when not used. Instead of leaving them running, consider shutting them off outside regular business hours.
  - Learn more about starting/stopping VMs during off-hours.

Utilize auto-scaling and bursting capabilities

It's not uncommon with oversized compute workloads where much of the capacity is never utilized, ultimately leading to a waste of energy.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Review auto-scaling guidance for Azure workloads.
- Review the B-series burstable virtual machine sizes.
- Consider that it may require tuning to prevent unnecessary scaling during short bursts of high demand, as opposed to a static increase in demand.
- Consider the application architecture as part of scaling considerations. For example, logical components should scale independently to match the demand of that component, as opposed to scaling the entire application if only a portion of the components needs scaling.

Match the scalability needs

Consider the platform and whether it meets the scalability needs of the solution. For example, having provisioned resources with a dedicated allocation may lead to unused or underutilized compute resources.

Examples:

- Provisioning an Azure App Service Environment (ASE) over an App Service plan may lead to having provisioned compute, whether utilized or not.
- Choosing the Azure API Management Premium tier instead of the consumption tier leads to unused resources if you aren't utilizing it fully.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Review the platform design decisions regarding scalability, and ensure the workload utilizes as much of the provisioned resources as possible.
- Consider this tradeoff: Some services require a higher tier to access certain features and capabilities regardless of resource utilization.
- Consider and prefer services that allow dynamic tier scaling where possible.

Evaluate Ampere Altra Arm-based processors for Virtual Machines

The Arm-based VMs represent a cost-effective and power-efficient option that doesn't compromise on the required performance.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Evaluate if the Ampere Altra Arm-based VMs is a good option for your workloads.
- Read more about Azure Virtual Machines with Ampere Altra Arm–based processors on Azure.

Delete zombie workloads

Consider discovering unutilized workloads and resources and if there are any orphaned resources in your subscriptions.

Green Software Foundation alignment: Hardware efficiency, Energy efficiency

Recommendation:

- Delete any orphaned workloads or resources if they're no longer necessary.

Next step

Review the design considerations for deployment and testing.

Deployment and testing

Testing considerations for sustainable workloads on Azure

Article • 10/28/2022

Organizations developing and deploying solutions to the cloud also need reliable testing. Learn about the considerations and recommendations for running workload tests and how to optimize for a more sustainable testing model.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Testing efficiency

Run integration, performance, load, or any other intense testing during low-carbon periods

Running integration, performance, load, or any other intense testing capability may result in much processing. A well-crafted design for testing the deployed workloads can help ensure full utilization of the available resources, reducing carbon emissions.

Green Software Foundation alignment: Carbon awareness

Recommendation:

- Where you have the data available to you, plan for running testing when the data center's energy mix primarily uses renewable energy. It may, for example, be more beneficial to run testing during the night in some regions.

Automate CI/CD to scale worker agents as needed

Running underutilized or inactive CI/CD agents results in more emissions.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Keeps the compute utilization high, based on the current demand, avoiding unnecessary capacity allocation.
- Only scale out when necessary, and when not testing, scale in. Ultimately this ensures there's no idle compute resources in test environments.
- Consider optimized platform services like containers over testing in a VM, utilizing the platform to reduce maintenance.

Consider caching when using CI/CD agents

Using caching mechanisms during CI/CD can reduce compute time and, thus, carbon emissions.

Green Software Foundation alignment: Energy Efficiency

Recommendation:

- Store results from steps in a cache and re-use them between different CI/CD runs when possible: when there are steps that take CPU time to produce an artifact that does not often change between different runs, it is wise to save it for future usage so that CPU time is not wasted on every run producing the same artifact, over and over.
- If the CI/CD agent is self-hosted, use a cache local to the agent to further reduce data transfers and emissions. This ensures that the cache is not transferred over the network, which can be a significant source of emissions.

Split large code repositories

Splitting large repositories can help the CI/CD phases, where only the parts of the code that have changed are compiled. This reduces compute time, which ultimately lowers carbon emissions.

Green Software Foundation alignment: Energy Efficiency

Recommendation:

- Split large code repositories, separating main code from libraries and dependencies.
- Publish and re-use artifacts and libraries of code that are common across multiple repositories.

Recommendation:

- Split large repositories of code into smaller ones, separating main code from libraries and dependencies.
- Publish and re-use artifacts and libraries of code that are common across multiple repositories.

Profiling and measuring

Measuring, profiling, and testing workloads are imperative to understanding how to best use allocated resources.

Assess where parallelization is possible

Without properly profiling and testing workloads, it's difficult to know if it's making the best use of the underlying platform and deployed resources.

Green Software Foundation alignment: *Measuring sustainability*

Recommendation:

- Test your applications to understand concurrent requests, simultaneous processing, and more.
- If you're running Machine Learning (ML) for tests, consider machines with a GPU for better efficiency gains.
- Identify if the workload is performance intensive and work toward optimization.
- *Consider this tradeoff:* Running GPU-based machines for ML tests may increase the cost.

Assess with chaos engineering

Running integration, performance and load tests increase the reliability of a workload. However, the introduction of chaos engineering can significantly help improve reliability and resilience and how the applications react to failures. In doing so, the workload can be optimized to handle failures gracefully and with less wasted resources.

Green Software Foundation alignment: *Measuring sustainability*

Recommendation:

- Use load testing or chaos engineering to assess how the workload handles platform outages and traffic spikes or dips. This helps increase service resilience and the ability to react to failures, allowing for a more optimized fault handling.

- *Consider this tradeoff:* Injecting fault during chaos engineering and increasing the load on any system also increases the emissions used for the testing resources. Evaluate how and when you can utilize chaos engineering to increase the workload reliability while considering the climate impact of running unnecessary testing sessions.
- Another angle to this is using chaos engineering to test energy faults or moments with higher carbon emissions: consider setting up tests that will challenge your application to consume the minimum possible energy. Define how the application will react to such conditions with a specific "eco" version informing users that they're emitting the minimum possible carbon by sacrificing some features and possibly some performance. This can also be your benchmark application for scoring its sustainability.

Establish CPU and Memory thresholds in testing

Help build tests for testing sustainability in your application. Consider having a baseline CPU utilization measurement, and detect abnormal changes to the CPU utilization baseline when tests run. With a baseline, suboptimal decisions made in recent code changes can be discovered earlier.

Adding tests and quality gates into the deployment and testing pipeline helps avoid deploying non-sustainable solutions, contributing to lowered emissions.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Monitor CPU and memory allocations when running integration tests or unit tests.
- Find abnormally high resource consumption areas in the application code and focus on mitigating those first.
- Configure alerts or test failures if surpassing the established baseline values, helping avoid deploying non-sustainable workloads.
- Consider this tradeoff: As applications grow, the baseline may need to shift accordingly to avoid failing the tests when introducing new features.

Next step

Review the design considerations for operational procedures.

Operational procedures

Operational procedure considerations for sustainable workloads on Azure

Article • 10/12/2022

The discipline of green software and its implementation within cloud efficiency patterns is relatively recent, and no specific and universal standards have been agreed upon yet.

The Green Software Foundation works on creating and standardizing ways of making green software. However, it's vital that everyone considers this aspect in their daily work and that when designing, planning, and deploying Azure workloads, we consider the best practices that are already available and prepare our environment to incorporate new standards when ready.

This document will guide you through setting up an environment for measuring and continuously improving your Azure workloads' cost and carbon efficiency.

> **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Measure and track carbon impact

To optimize or improve something, we first must decide what we want to change and how to measure it. In this section, you'll learn about best practices and guidelines to measure and track the sustainability impact of your workloads.

The Emissions Impact Dashboard

An essential aspect of working toward any sustainability goal is tracking and quantifying progress. If you can't track and measure the impact, you'll never be sure if the efforts are worthwhile. The Emissions Impact Dashboard is a Power BI dashboard that will give you a measure of the carbon impact of all your services and resource groups in your Azure subscription(s).

The Emissions Impact Dashboards produce insights in various forms, and allows for a wide range of reporting capabilities:

- Series of visual representations in the dashboard itself.
- Snapshot export to Excel, Power Point and PDF.
- Continuous export to Microsoft Sustainability Manager and Dataverse.

Green Software Foundation alignment: Measuring sustainability

Recommendation:

- Use the Emissions Impact Dashboard to record current and future environmental impact.
- Identify and track metrics to quantify the achievement of technical, business, and sustainability outcomes.
- Rely on tooling to help measure the impact, and record any changes made to your workload.
- Learn more about the Sustainability and Dataverse API access in the Microsoft Learn module Access Microsoft Sustainability Manager data.

Define emissions target

The Software Carbon Intensity (SCI) is the score you're looking for to measure the carbon impact of your application(s) by adding the scalability and cost metrics to any carbon emissions measurement.

If you aren't using the Emissions Impact Dashboard, there are still ways of building carbon proxies that allow you to measure your application's impact on emissions.

It can be a challenge to build carbon proxies for existing applications. Therefore, we recommend planning for efficiency targets during the design phase of every workload. When adding new workloads to Azure, you should consider planning for costs and emissions that will add to your existing footprint. The main goal should always be not to emit carbon, so ideally, you should immediately find an optimization pattern to make up for the new emissions.

The next step is to define your target emissions, either for a single application or for your entire set of cloud workloads. The target can also include cost constraints, making it even easier to build upon since shrinking costs will give you some budget to optimize emissions. Once you know your target, the cloud efficiency continuous optimization process can start.

Green Software Foundation alignment: Measuring sustainability

Recommendations:

- Calculate your new workload's minimum cost and carbon emissions (where applicable).
- Track progress with Service Level Objectives (SLO), Service Level Agreements (SLA), or other performance metrics.
- Provide optimization patterns to accommodate the new application to your overall cloud efficiency score.

Identify the metrics and set improvement goals

Once you've defined your target, you'll need to identify a few metrics that you can measure to prove your changes had a positive effect on efficiency.

The metrics can, as an example, be derived from these categories:

- Application performance metrics.
- Cost optimization metrics.
- Carbon emissions metrics (or proxies).

Green Software Foundation alignment: Measuring sustainability

Recommendation:

- Discuss with every application owner since the impact of optimizing can vary and might affect many users.
- Make sure that any plan that impacts performance is agreed upon and communicated clearly to the app users so that they know that a lower performance may be necessary for the greater good of fewer carbon emissions.
- If you've connected the Microsoft Emissions Impact Dashboard (EID) to your Microsoft Sustainability Manager (MSM) instance, you can use the Goal Tracking feature in MSM to define and track your goals by linking them to live data from EID.

Cost optimization as a proxy

Sometimes the ease of deploying cloud resources makes us forget what is useful and what is simply a waste of resources, money, and carbon. The message here's that experiments in the cloud can sometimes be costly in terms of overall cloud efficiency, not purely cost, while bringing no innovation.

Use cloud resources wisely, considering any extra workload's carbon footprint.

When defining your SCI, you can use carbon proxies to compensate for the lack of specific standards and measurements. One of the safest and most potent proxies for carbon emissions are your application(s) cost. Reducing unnecessary spending lowers the number of excessive emissions from deployed workloads as you're using fewer cloud resources.

Linking cost performance metrics to carbon efficiency can be a sound strategy because you won't necessarily need to compromise on your defined workload Key Performance Indicators (KPI) by optimizing cost and reducing carbon emissions. However, you might decide that you're prepared to sacrifice a KPI towards your carbon goal, which can also be part of your strategy.

Green Software Foundation alignment: Measuring sustainability

Recommendation:

- Review the concept of using a proxy solution to measure emissions.
- Leverage the guidance in the Azure Well-Architected Framework Cost Optimization pillar.

Defining policies

Azure Policy is a powerful tool that can make some decisions for your cloud efficiency easier to implement. Consider defining one of more policies to keep your Azure virtual data center continuously optimized.

Green Software Foundation alignment: Climate commitments

Recommendation:

- Incorporate and use the cost policies available in the Cloud Adoption Framework.
- Leverage built-in policies relevant to cost in Azure Policy, as they're technically closely tied to sustainability.
- Customize Azure Policy policies according to green software principles. For example, create a new Azure Policy initiative for "Sustainability".
  - Consider this tradeoff: Enforcement of new policies must not impact any unplanned operational performance metric.

Community and knowledge sharing

Teams needs to be constantly aware of new advancements in sustainability, so they leverage these learnings when implementing workloads.

Building a community around cloud efficiency and green software is a good starting point to foster cloud efficiency awareness and culture across your organization.

Create a sustainability community

Creating a sustainability community doesn't have to be a tedious task. Start with a small team that will invest some time in learning the sustainability status and the relevant information on green software. This team can also join the Green Software Foundation and be part of the teams that create rules, standards, and more.

The Core cloud Efficiency team will have to be up to date with all the innovative tools and principles that drive your Azure workload's cost and carbon footprint.

Green Software Foundation alignment: Climate commitments

Recommendation:

- Define policies and targets, and communicate their efforts and goals with the rest of the organization.
- Learn more by reading how do I start a sustainability community in my organization?

Plan for learning

Make time for the core team to learn about advancements in sustainable operations. Meanwhile, ensure that your entire organization starts thinking about green software and how to contribute to the sustainability picture with their daily choices.

Green Software Foundation alignment: Climate commitments

Recommendation:

Review these popular training and learning resources:

- Use the self-paced learning module to Learn about The Principles of Sustainable Software Engineering.
- Use the self-paced learning path to Get started with Microsoft Cloud for Sustainability.
- Find more resources in the Microsoft Sustainability Learning Center.

Share best practices across teams

Driving adoption of sustainability efforts requires input and work from across the organization.

Green Software Foundation alignment: Climate commitments

Recommendation:

- Let team members share their workload and company-specific best practices for sustainable operations.
- Set up a shared repository of best practices and guidance that have been tested in your environment with tangible results.
- Consider frequent knowledge-sharing sessions or internal webinars for getting everyone up to speed.

Plan for incentives

The quickest way of enforcing policies and creating the right culture is by setting incentives for improving the environmental sustainability of a workload by either putting sustainability as a core KPI or adding it to the overall efficiency of the applications.

Many software partners already include green software in their best practices. Therefore, ensure that your efficiency targets are defined and accepted when discussing the workload.

Green Software Foundation alignment: Climate commitments

Recommendations:

- Promote carbon-aware applications. Reward application owners if the measured carbon footprint meets the KPI.
- Introduce gamification by creating a friendly culture of sustainability competition —track records to promote green workloads, SCI scoring, and any optimization or improvement on the score.
- Consider introducing loyalty programs, where participants get incentives when they can prove the cloud efficiency of their applications.
- Explore the opportunity to introduce badges like "Carbon Aware" and "Carbon Optimized".

Next step

Review the design considerations for networking and connectivity.

Networking and connectivity

Networking considerations for sustainable workloads on Azure

Article • 10/12/2022

Most workloads in the cloud rely heavily on networking to operate. Whether internal networking or public-facing workloads, the components and services used in provisioned solutions must consider the impact of carbon emissions. Consider that network equipment consumes electricity, including traffic between the data centers and end consumers. Learn about considerations and recommendations to enhance and optimize network efficiency to reduce unnecessary carbon emissions.

Internet traversal between data centers and end consumers is a significant Scope 3 emission. Therefore, recommendations in this section are aligned with the Principles of Green Software Networking area to improve networking efficiency.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Network efficiency

Reduce unnecessary network traffic and lower bandwidth requirements where possible, allowing for a more optimized network efficiency with less carbon emission.

Make use of a CDN

Unnecessary traffic on the network should be avoided, as it's a cause for extra carbon emissions.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- A CDN helps minimize latency through storing frequently read static data closer to consumers, and helps reduce the network traversal and server load.
- Ensure to follow best practices for CDN.

Follow caching best practices

Minimizing the amount of data transferred is crucial.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- Caching is a well-understood design technique to improve performance and efficiency.
- A caching solution helps reduce network traversal and reduces the server load.
- Consider that it may require tuning of parameters to maximize the benefit and minimize the carbon drawbacks. For example, setting a Time to Live (TTL).
- Adding in-memory caching can help use idle compute resources, increasing the compute density of resources that are already allocated.
- Read caching best practices.

Select Azure regions based on where the customer resides

The location of an application's consumers can be disparate, and it can be challenging to serve requests with good performance and energy efficiency if the distance is too great.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Deploy or move Azure resources across regions to better serve the applications from where most consumers reside.

Use managed audio and video streaming services with built-in compression

Applications making use of a media streaming service may have high requirements for bandwidth and compression, and can have a substantial carbon footprint if not designed carefully.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- By making use of a managed service for audio and video, applications can leverage built-in optimizations like encoding, compressions, and more.
- Read about managed audio and video streaming services.

Enable network file compression

Networks sending uncompressed data can have a higher requirement on bandwidth, the allocated resources, and the solution in general. Consider compressing data to optimize the workload and design for a more network efficient solution.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Reduce the network payload by improving CDN performance.

Maximize network utilization within the same cloud and region

Operating solutions in multiple regions have a networking impact. Network traversals between components in Azure are optimized to stay within the Azure infrastructure. However, any network traffic destined for the internet or a component in another cloud involves the public internet's router resources, which you have no control over regarding resource impact measurement or utilization.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Keeping resources in a single cloud gives you maximum control and allows the cloud provider to optimize the network routing.
- Maximize network utilization within the same cloud and, if possible, within the same region.
- Since the cost can be a proxy for sustainability, review the Azure regions documentation in the Cost Optimization pillar of the Azure Well-Architected Framework.

Next step

Review the design considerations for storage.

Storage

Data and storage design considerations for sustainable workloads on Azure

Article • 10/12/2022

Data storage in Azure is a crucial component of most provisioned workloads. Learn how to design for a more sustainable data storage architecture and optimize existing deployments.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Storage efficiency

Build solutions with efficient storage to increase performance, lower the required bandwidth, and minimize unnecessary storage design climate impact.

Enable storage compression

Storing much uncompressed data can result in unnecessary bandwidth waste and increase the storage capacity requirements.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- A solution to reduce the storage requirements, including both capacity and required bandwidth to write or retrieve data. For example, compressing files in Azure Front Door and compressing files in Azure CDN.
- Compression is a well-known design technique to improve network performance.
- Consider the tradeoff of compression: Does the benefit of compression outweigh the increased *carbon* cost in the resources (CPU, RAM) needed to perform the compression/decompression?

Optimize database query performance

Querying extensive databases or retrieving much information simultaneously can have a performance penalty. Ideally, apps should optimize for query performance.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Reduces the latency of data retrieval while also reducing the load on the database.
- Understand the query performance for Azure SQL Databases
- There are many well-known ways to optimize data query performance, for example tuning apps and databases for performance in an Azure SQL database.
- Consider that it may require fine-tuning to achieve optimal results.

Use the best suited storage access tier

The carbon impact of data retrieved from hot storage can be higher than data from cold- or archive storage. Designing solutions with the correct data access pattern can enhance the application's carbon efficiency.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Use storage best suited for the application's data access patterns.
- Make sure your most frequent data is stored in hot storage, making it easy to retrieve and doesn't require more processing to access.
- Infrequently used data should be stored in cold or offline archive storage, using less energy.

Only store what is relevant

Backup is a crucial part of reliability. However, storing backups indefinitely can quickly allocate much unnecessary disk space. Consider how you plan backup storage retention.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Implement policies to streamline the process of storing and keeping relevant information. Microsoft Purview can help label data and add time-based purging to delete it after a retention period automatically. Additionally, this lets you stay in control of your data and reduces the amount of data to process and transfer.
- Workloads integrated with Azure Monitor can rely on Data Collection Rules (DCR) to specify what data should be collected, how to transform that data, and where to

send the data.

Determine the most suitable access tier for blob data

Consider whether to store data in an online tier or an offline tier. Online tiers are optimized for storing data that is accessed or modified frequently. Offline tiers are optimized for storing data that is rarely accessed.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Read Hot, Cool, and Archive access tiers for blob data.

Reduce the number of recovery points for VM backups

Recovery points aren't automatically cleaned up. Therefore, consider where soft delete is enabled for Azure Backup. The expired recovery points aren't cleaned up automatically.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Read more about the impact of expired recovery points for items in soft deleted state.

Revise backup and retention policies

Consider reviewing backup policies and retention periods for backups to avoid storing unnecessary data.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Review and revise backup and retention policies to minimize storage overhead.
- Actively review and delete backups that are no longer needed.

Optimize the collection of logs

Continuously collecting logs across workloads can quickly aggregate and store lots of unused data.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Make sure you are logging and retaining only data that is relevant to your needs.
- Read more about the Cost optimization and Log Analytics.

Next step

Review the design considerations for security.

Security

Security considerations for sustainable workloads on Azure

Article • 10/12/2022

Designing sustainable workloads on Azure must encompass security, which is a foundational principle through all phases of a project. Learn about considerations and recommendations leading to a more sustainable security posture.

> ⓘ **Important**
>
> This article is part of the **Azure Well-Architected sustainable workload** series. If you aren't familiar with this series, we recommend you start with **what is a sustainable workload?**

Security monitoring

Use cloud native security monitoring solutions to optimize for sustainability.

Use cloud native log collection methods where applicable

Traditionally, log collection methods for ingestion to a Security Information and Event Management (SIEM) solution required the use of an intermediary resource to collect, parse, filter and transmit logs onward to the central collection system. Using this design can carry an overhead with more infrastructure and associated financial and carbon-related costs.

Green Software Foundation alignment: Hardware efficiency, Energy efficiency

Recommendation:

- Using cloud native service-to-service connectors simplify the integration between the services and the SIEM, and removes the overhead of extra infrastructure.
- It's possible to ingest log data from existing compute resources using previously deployed agents such as the Azure Monitor Analytics agent. Review how to migrate to Azure Monitor agent from Log Analytics agent.
- Consider this tradeoff: Deploying more monitoring agents will increase the overhead in processing as it needs more compute resources. Carefully design and plan for how much information is needed to cover the security requirements of the solution and find a suitable level of information to store and keep.

- A possible solution to reduce unnecessary data collection is to rely on the Azure Monitor Data Collection Rules (DCR).

Avoid transferring large unfiltered data sets from one cloud service provider to another

Conventional SIEM solutions required all log data to be ingested and stored in a centralized location. In a multicloud environment, this solution can lead to a large amount of data being transferred out of a cloud service provide and into another, causing increased burden on the network and storage infrastructure.

Green Software Foundation alignment: Carbon efficiency, Energy efficiency

Recommendation:

- Cloud native security services can perform localized analysis on relevant security data source. This analysis allows the bulk of log data to remain within the source cloud service provider environment. Cloud native SIEM solutions can be connected via an API or connector to these security services to transmit only the relevant security incident or event data. This solution can greatly reduce the amount of data transferred while maintaining a high level of security information to respond to an incident.

In time, using the described approach helps reduce data egress and storage costs, which inherently help reduce emissions.

Filter or exclude log sources before transmission or ingestion into a SIEM

Consider the complexity and cost of storing all logs from all possible sources. For instance, applications, servers, diagnostics and platform activity.

Green Software Foundation alignment: Carbon efficiency, Energy efficiency

Recommendation:

- When designing a log collection strategy for cloud native SIEM solutions, consider the use cases based on the Microsoft Sentinel analytics rules required for your environment and match up the required log sources to support those rules.
- This option can help remove the unnecessary transmission and storage of log data, reducing the carbon emissions on the environment.

Archive log data to long-term storage

Many customers have a requirement to store log data for an extended period due to regulatory compliance reasons. In these cases, storing log data in the primary storage location of the SIEM system is a costly solution.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Log data can be moved out to a cheaper long-term storage option which respects the retention policies of the customer, but lowers the cost by utilizing separate storage locations.

Network architecture

Increase the efficiency and avoid unnecessary traffic by following good practices for network security architectures.

Use cloud native network security controls to eliminate unnecessary network traffic

When you use a centralized routing- and firewall design, all network traffic is sent to the hub for inspection, filtering, and onward routing. While this approach centralizes policy enforcement, it can create an overhead on the network of unnecessary traffic from the source resources.

Green Software Foundation alignment: Hardware efficiency, Energy efficiency

Recommendation:

- Use Network security groups and Application security groups to help filter traffic at the source, and to remove the unnecessary data transmission. Using these capabilities can help reduce the burden on the cloud infrastructure, with lower bandwidth requirements and less infrastructure to own and manage.

Minimize routing from endpoints to the destination

In many customer environments, especially in hybrid deployments, all end user device network traffic is routed through on-premises systems before being allowed to reach the internet. Usually, this happens due to the requirement to inspect all internet traffic.

Often, this requires higher capacity network security appliances within the on-premises environment, or more appliances within the cloud environment.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Minimize routing from endpoints to the destination.
  - Where possible, end user devices should be optimized to split out known traffic directly to cloud services while continuing to route and inspect traffic for all other destinations. Bringing these capabilities and policies closer to the end user device prevents unnecessary network traffic and its associated overhead.

Use network security tools with auto-scaling capabilities

Based on network traffic, there will be times when demand of the security appliance will be high, and other times where it will be lower. Many network security appliances are deployed to a scale to cope with the highest expected demand, leading to inefficiencies. Additionally, reconfiguration of these tools often requires a reboot leading to unacceptable downtime and management overhead.

Green Software Foundation alignment: Hardware efficiency

Recommendation:

- Making use of auto-scaling allows the rightsizing of the backend resources to meet demand without manual intervention.
- This approach will vastly reduce the time to react to network traffic changes, resulting in a reduced waste of unnecessary resources, and increases your sustainability effect.
- Learn more about relevant services by reading how to enable a Web Application Firewall (WAF) on an Application Gateway, and deploy and configure Azure Firewall Premium.

Evaluate whether to use TLS termination

Terminating and re-establishing TLS is CPU consumption that might be unnecessary in certain architectures.

Green Software Foundation alignment: Energy efficiency

Recommendation:

- Consider if you can terminate TLS at your border gateway and continue with non-TLS to your workload load balancer and onwards to your workload.
- Review the information on TLS termination to better understand the performance and utilization impact it offers.
- Consider the tradeoff: A balanced level of security can offer a more sustainable and energy efficient workload while a higher level of security may increase the requirements on compute resources.

Use DDoS protection

Distributed Denial of Service (DDoS) attacks aim to disrupt operational systems by overwhelming them, creating a significant impact on the resources in the cloud. Successful attacks flood network and compute resources, leading to an unnecessary spike in usage and cost.

Green Software Foundation alignment: Energy efficiency, Hardware efficiency

Recommendation:

- DDoS protection seeks to mitigate attacks at an abstracted layer, so the attack is mitigated before reaching any customer operated services.
  - Mitigating any malicious usage of compute and network services will ultimately help reduce unnecessary carbon emissions.

Endpoint security

It's imperative that we secure our workloads and solutions in the cloud. Understanding how we can optimize our mitigation tactics all the way down to the client devices can have a positive outcome for reducing emissions.

Integrate Microsoft Defender for Endpoint

Many attacks on cloud infrastructure seek to misuse deployed resources for the attacker's direct gain. Two such misuse cases are botnets and crypto mining.

Both of these cases involve taking control of customer-operated compute resources and use them to either create new cryptocurrency coins, or as a network of resources from which to launch a secondary action like a DDoS attack, or mass e-mail spam campaigns.

Green Software Foundation alignment: Hardware efficiency

Recommendations:

- Integrate Microsoft Defender for Endpoint with Defender for Cloud to identify and shut down crypto mining and botnets.
  - The EDR capabilities provide advanced attack detections and are able to take response actions to remediate those threats. The unnecessary resource usage created by these common attacks can quickly be discovered and remediated, often without the intervention of a security analyst.

Reporting

Getting the right information and insights at the right time is important for producing reports around emissions from your security appliances.

Tag security resources

It can be a challenge to quickly find and report on all security appliances in your tenant. Identifying the security resources can help when designing a strategy for a more sustainable operating model for your business.

Green Software Foundation alignment: Measuring sustainability

Recommendation:

- Tag security resources to record emissions impact of security resources.

Next step

Review the design principles for sustainability.

Design principles

Azure Well-Architected Framework review - Azure Service Fabric

Article • 01/11/2023

Azure Service Fabric is a distributed systems platform that makes it easy to package, deploy, and manage scalable and reliable microservices and containers. These resources are deployed onto a network-connected set of virtual or physical machines, which is called a **cluster**.

There are two clusters models in Azure Service Fabric: **standard clusters** and **managed clusters**.

Standard clusters require you to define a cluster resource alongside a number of supporting resources. These resources must be set up correctly upon deployment and maintained correctly throughout the lifecycle of the cluster. Otherwise, the cluster and your services will not function properly.

Managed clusters simplify your deployment and management operations. The managed cluster model consists of a single Service Fabric managed cluster resource that encapsulates and abstracts away the underlying resources.

This article primarily discusses the **managed cluster** model for simplicity. However, call-outs are made for any special considerations that apply to the **standard cluster** model.

In this article, you learn architectural best practices for Azure Service Fabric. The guidance is based on the five pillars of architectural excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

Prerequisites

- Understanding the Well-Architected Framework pillars can help produce a high quality, stable, and efficient cloud architecture. Check out the Azure Well-Architected Framework overview page to review the five pillars of architectural excellence.

- Reviewing the core concepts of Azure Service Fabric and microservice architecture can help you understand the context of the best practices provided in this article.

Reliability

The following sections cover design considerations and configuration recommendations, specific to Azure Service Fabric and reliability.

When discussing reliability with Azure Service Fabric, it's important to distinguish between *cluster reliability* and *workload reliability*. Cluster reliability is a shared responsibility between the Service Fabric cluster admin and their resource provider, while workload reliability is the domain of a developer. Azure Service Fabric has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

For more information about Azure Service Fabric cluster reliability, check out the capacity planning documentation.

For more information about Azure Service Fabric workload reliability, reference the Reliability subsystem included in the Service Fabric architecture.

Design checklist

As you make design choices for Azure Service Fabric, review the design principles for adding reliability to the architecture.

- ✓ **Cluster architecture:** Use Standard SKU for production scenarios. **Standard cluster:** Use durability level Silver (5 VMs) or greater for production scenarios.
- ✓ **Cluster architecture:** For critical workloads, consider using Availability Zones for your Service Fabric clusters.
- ✓ **Cluster architecture:** For production scenarios, use the Standard tier load balancer. Managed clusters create an Azure public Standard Load Balancer and fully qualified domain name with a static public IP for both the primary and secondary node types. You can also bring your own load balancer, which supports both Basic and Standard SKU load balancers.
- ✓ **Cluster architecture:** Create additional, secondary node types for your workloads.

Recommendations

Explore the following table of recommendations to optimize your Azure Service Fabric configuration for service reliability:

Azure Service Fabric Recommendation	Benefit
Cluster architecture: Use Standard SKU for production scenarios.	This level ensures the resource provider maintains cluster reliability. **Standard cluster:** A Standard SKU managed cluster provides the equivalent of durability level Silver. To achieve this using the standard cluster model, you will need to use 5 VMs (or more).
Cluster architecture: Consider using Availability Zones for your Service Fabric clusters.	Service Fabric managed cluster supports deployments that span across multiple Availability Zones to provide zone resiliency. This configuration will ensure high-availability of the critical system services and your applications to protect from single-points-of-failure.
Cluster architecture: Consider using Azure API Management to expose and offload cross-cutting functionality for APIs hosted on the cluster.	API Management can integrate with Service Fabric directly.
Workload architecture: For stateful workload scenarios, consider using Reliable Services.	The Reliable Services model allows your services to stay up even in unreliable environments where your machines fail or hit network issues, or in cases where the services themselves encounter errors and crash or fail. For stateful services, your state is preserved even in the presence of network or other failures.

For more suggestions, see Principles of the reliability pillar.

Security

The following sections cover design considerations and configuration recommendations, specific to Azure Service Fabric and security.

When discussing security with Azure Service Fabric, it's important to distinguish between *cluster security* and *workload security*. Cluster security is a shared responsibility between the Service Fabric cluster admin and their resource provider, while workload security is the domain of a developer. Azure Service Fabric has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

For more information about Azure Service Fabric cluster security, check out Service Fabric cluster security scenarios.

For more information about Azure Service Fabric workload security, reference Service Fabric application and service security.

Design checklist

As you make design choices for Azure Service Fabric, review the design principles for adding security to the architecture.

- ✓ **Cluster architecture:** Ensure Network Security Groups (NSG) are configured to restrict traffic flow between subnets and node types. Ensure that the correct ports are opened for application deployment and workloads.
- ✓ **Cluster architecture:** When using the Service Fabric Secret Store to distribute secrets, use a separate data encipherment certificate to encrypt the values.
- ✓ **Cluster architecture:** Deploy client certificates by adding them to Azure Key Vault and referencing the URI in your deployment.
- ✓ **Cluster architecture:** Enable Azure Active Directory integration for your cluster to ensure users can access Service Fabric Explorer using their Azure Active Directory (AAD) credentials. Don't distribute the cluster client certificates among users to access Explorer.
- ✓ **Cluster architecture:** For client authentication, use admin and read-only client certificates and/or AAD authentication.
- ✓ **Cluster and workload architectures:** Create a process for monitoring the expiration date of client certificates.
- ✓ **Cluster and workload architectures:** Maintain separate clusters for development, staging, and production.

Recommendations

Consider the following recommendations to optimize your Azure Service Fabric configuration for security:

Azure Service Fabric Recommendation	Benefit
Cluster architecture: Ensure Network Security Groups (NSG) are configured to restrict traffic flow between subnets and node types.	For example, you may have an API Management instance (one subnet), a frontend subnet (exposing a website directly), and a backend subnet (accessible only to frontend).

Azure Service Fabric Recommendation	Benefit
Cluster architecture: Deploy Key Vault certificates to Service Fabric cluster virtual machine scale sets.	Centralizing storage of application secrets in Azure Key Vault allows you to control their distribution. Key Vault greatly reduces the chances that secrets may be accidentally leaked.
Cluster architecture: Apply an Access Control List (ACL) to your client certificate for your Service Fabric cluster.	Using an ACL provides an additional level of authentication.
Cluster architecture: Use resource requests and limits to govern resource usage across the nodes in your cluster.	Enforcing resource limits helps ensure that one service doesn't consume too many resources and starve other services.
Workload architecture: Encrypt Service Fabric package secret values.	Encryption on your secret values provides an additional level of security.
Workload architecture: Include client certificates in Service Fabric applications.	Having your applications use client certificates for authentication provides opportunities for security at both the cluster and workload level.
Workload architecture: Authenticate Service Fabric applications to Azure Resources using Managed Identity.	Using Managed Identity allow you to securely manage the credentials in your code for authenticating to various services without saving them locally on a developer workstation or in source control.
Cluster and workload architectures: Follow Service Fabric best practices when hosting untrusted applications.	Following the best practices provides a security standard to follow.

For more suggestions, see Principles of the security pillar.

Azure Advisor helps you ensure and improve the security of Azure Service Fabric. You can review the recommendations in the Azure Advisor section of this article.

Policy definitions

Azure Policy helps maintain organizational standards and assess compliance across your resources. Keep the following built-in policies in mind as you configure Azure Service Fabric:

- Service Fabric clusters should have the ClusterProtectionLevel property set to `EncryptAndSign`. This is the default value for managed clusters and isn't

changeable. **Standard cluster:** Ensure you set ClusterProtectionLevel to `EncryptAndSign`.
- Service Fabric clusters should only use Azure Active Directory for client authentication.

All built-in policy definitions related to Azure Service Fabric are listed in Built-in policies - Service Fabric.

Cost optimization

The following sections cover design considerations and configuration recommendations, specific to Azure Service Fabric and cost optimization.

When discussing cost optimization with Azure Service Fabric, it's important to distinguish between *cost of cluster resources* and *cost of workload resources*. Cluster resources are a shared responsibility between the Service Fabric cluster admin and their resource provider, while workload resources are the domain of a developer. Azure Service Fabric has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

For cluster cost optimization, go to the Azure pricing calculator and select **Azure Service Fabric** from the available products. You can test different configuration and payment plans in the calculator.

For more information about Azure Service Fabric workload pricing, check out the example cost calculation process for application planning.

Design checklist

As you make design choices for Azure Service Fabric, review the design principles for optimizing the cost of your architecture.

- ✓ **Cluster architecture:** Select appropriate VM SKU.
- ✓ **Cluster architecture:** Use appropriate node type and size.
- ✓ **Cluster and workload architectures:** Use appropriate managed disk tier and size.

Recommendations

Explore the following table of recommendations to optimize your Azure Service Fabric configuration for cost:

Azure Service Fabric Recommendation	Benefit
Cluster architecture: Avoid VM SKUs with temp disk offerings.	Service Fabric uses managed disks by default, so avoiding temp disk offerings ensures you don't pay for unneeded resources.
Cluster architecture: If you need to select a certain VM SKU for capacity reasons and it happens to offer temp disk, consider using temporary disk support for your stateless workloads.	Make the most of the resources you're paying for. Using a temporary disk instead of a managed disk can reduce costs for stateless workloads.
Cluster and workload architectures: Align SKU selection and managed disk size with workload requirements.	Matching your selection to your workload demands ensures you don't pay for unneeded resources.

For more suggestions, see Principles of the cost optimization pillar.

Operational excellence

The following sections cover design considerations and configuration recommendations, specific to Azure Service Fabric and operational excellence.

When discussing security with Azure Service Fabric, it's important to distinguish between *cluster operation* and *workload operation*. Cluster operation is a shared responsibility between the Service Fabric cluster admin and their resource provider, while workload operation is the domain of a developer. Azure Service Fabric has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

Design checklist

As you make design choices for Azure Service Fabric, review the design principles for operational excellence.

- ✓ **Cluster architecture:** Prepare a cluster monitoring solution.
- ✓ **Cluster architecture:** Review the cluster health policies in the Service Fabric health model.

- ✓ **Workload architecture:** Prepare an application monitoring solution.
- ✓ **Workload architecture:** Review the application and service type health policies in the Service Fabric health model.
- ✓ **Cluster and workload architectures:** Prepare an infrastructure monitoring solution.
- ✓ **Cluster and workload architectures:** Design your cluster with build and release pipelines for continuous integration and deployment.

Recommendations

Explore the following table of recommendations to optimize your Azure Service Fabric configuration for operational excellence:

Azure Service Fabric Recommendation	Benefit
Workload architecture: Use Application Insights to monitor your workloads.	Application Insights integrates with the Azure platform, including Service Fabric.
Cluster and workload architectures: Create a process for monitoring the expiration date of client certificates.	For example, Key Vault offers a feature that sends an email when `x%` of the certificate's lifespan has elapsed.
Cluster and workload architectures: For pre-production clusters use Azure Chaos Studio to drill service disruption on a Virtual Machine Scale Set instance failure.	Practicing service disruption scenarios will help you understand what is at-risk in your infrastructure and how to best mitigate the issues if they arise.
Cluster and workload architectures: Use Azure Monitor to monitor cluster and container infrastructure events.	Azure Monitor integrates well with the Azure platform, including Service Fabric.
Cluster and workload architectures: Use Azure Pipelines for your continuous integration and deployment solution.	Azure Pipelines integrates well with the Azure platform, including Service Fabric.

For more suggestions, see Principles of the operational excellence pillar.

Performance efficiency

The following section covers configuration recommendations, specific to Azure Service Fabric and performance efficiency.

When discussing security with Azure Service Fabric, it's important to distinguish between *cluster operation* and *workload operation*. Cluster performance is a shared responsibility between the Service Fabric cluster admin and their resource provider, while workload performance is the domain of a developer. Azure Service Fabric has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

For more information about how Azure Service Fabric can reduce performance issues for your workload with Service Fabric performance counters, reference Monitoring and diagnostic best practices for Azure Service Fabric.

Design checklist

- ✓ **Cluster architecture:** Exclude the Service Fabric processes from Windows Defender to improve performance.
- ✓ **Cluster architecture:** Select appropriate VM SKU.
- ✓ **Workload architecture:** Decide what programming model you will use for your services.
- ✓ **Cluster and workload architectures:** Use appropriate managed disk tier and size.

Recommendations

Consider the following recommendations to optimize your Azure Service Fabric configuration for performance efficiency:

Azure Service Fabric Recommendation	Benefit
Cluster architecture: Exclude the Service Fabric processes from Windows Defender to improve performance.	By default, Windows Defender antivirus is installed on Windows Server 2016 and 2019. To reduce any performance impact and resource consumption overhead incurred by Windows Defender, and if your security policies allow you to exclude processes and paths for open-source software, you can exclude.
Cluster architecture: Consider using Autoscaling for your cluster.	Autoscaling gives great elasticity and enables addition or reduction of nodes on demand on a secondary node type. This automated and elastic behavior reduces the management overhead and potential business impact by monitoring and optimizing the amount of nodes servicing your workload.
Cluster architecture: Consider using Accelerated Networking.	Accelerated networking enables a high-performance path that bypasses the host from the data path, which reduces latency, jitter, and CPU utilization for the most demanding network workloads.

Azure Service Fabric Recommendation	Benefit
Cluster architecture: Considering using encryption at host instead of Azure Disk Encryption (ADE).	This encryption method improves on ADE by supporting all OS types and images, including custom images, for your VMs by encrypting data in the Azure Storage service.
Workload architecture: Review the Service Fabric programming models to decide what model would best suit your services.	Service Fabric supports several programming models. Each come with their own advantages and disadvantages. Knowing about the available programming models can help you make the best choices for designing your services.
Workload architecture: Leverage loosely-coupled microservices for your workloads where appropriate.	Using microservices allows you to get the most out of Service Fabric's features.
Workload architecture: Leverage event-driven architecture for your workloads where appropriate.	Using event-driven architecture allows you to get the most out of Service Fabric's features.
Workload architecture: Leverage background processing for your workloads where appropriate.	Using background processing allows you to get the most out of Service Fabric's features.
Cluster and workload architectures: Review the different ways you can scale your solution in Service Fabric.	You can use scaling to enable maximum resource utilization for your solution.

For more suggestions, see Principles of the performance efficiency pillar.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence when using Azure Service Fabric.

Security

- Service Fabric clusters should have the ClusterProtectionLevel property set to `EncryptAndSign`. This is the default value for managed clusters and isn't changeable. **Standard cluster:** Ensure you set ClusterProtectionLevel to `EncryptAndSign`.
- Service Fabric clusters should only use Azure Active Directory for client authentication.

Additional resources

Check out the Azure Service Fabric managed cluster configuration options article for a list of all the options you have while creating and maintaining your cluster.

Review the Azure application architecture fundamentals for guidance on how to develop your workloads. While Service Fabric can be used solely as a container hosting platform, using well-architected workloads leverages Service Fabric's full functionality.

Next steps

Use these recommendations as you create your Service Fabric managed cluster using an ARM template or through the Azure portal:

- Quickstart: Deploy a Service Fabric managed cluster with an Azure Resource Manager template
- Quickstart: Deploy a Service Fabric managed cluster using the Azure portal

Azure App Service and reliability

Article • 11/30/2022

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. This service adds the power of Microsoft Azure to your application, such as:

- Security
- Load balancing
- Autoscaling
- Automated management

To explore how Azure App Service can bolster the resiliency of your application workload, reference key features in Why use App Service?

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure App Service.

Design considerations

Microsoft guarantees that Azure App Service will be available `99.95%` of the time. However, no SLA is provided using either the Free or Shared tiers.

For more information, reference the SLA for App Service.

Checklist

Have you configured Azure App Service with resiliency in mind?

- ✓ Consider disabling ARR Affinity for your App Service.
- ✓ Use a different store for session state.
- ✓ Use Web Jobs.
- ✓ Enable Always On to ensure Web Jobs run reliably.
- ✓ Access the on-prem database using private connections like Azure VPN or Express Route.
- ✓ Set up backup and restore.
- ✓ Understand IP Address deprecation impact.
- ✓ Ensure App Service Environments (ASE) are deployed in highly available configurations across Availability Zones.
- ✓ Ensure the ASE Network is configured correctly.
- ✓ Consider configuring Upgrade preference if multiple environments are used.

- ✓ Plan for scaling out the ASE cluster.
- ✓ Use Deployment slots for resilient code deployments.
- ✓ Avoid unnecessary worker restarts.
- ✓ Use Run From Package to avoid deployment conflicts.
- ✓ Use Basic or higher plans with two or more worker instances for high availability.
- ✓ Evaluate the use of TCP and SNAT ports to avoid outbound connection errors.
- ✓ Enable Health check to identify non-responsive workers.
- ✓ Enable Autoscale to ensure adequate resources are available to service requests.
- ✓ Enable Local Cache to reduce dependencies on cluster file servers.
- ✓ Enable Diagnostic Logging to provide insight into application behavior.
- ✓ Enable Application Insights Alerts to signal fault conditions.
- ✓ Review Azure App Service diagnostics to ensure common problems are addressed.
- ✓ Evaluate per-app scaling for high density hosting on Azure App Service.

Configuration recommendations

Explore the following table of recommendations to optimize your App Service configuration for service reliability:

ASE Recommendation	Description
Consider disabling ARR Affinity for your App Service.	ARR Affinity (Application Request Routing) sets an affinity cookie, which is used to redirect users to the same node that handled their previous requests.
Use a different store for session state.	Storing session state in memory can result in losing session state when there's a problem with the application or App Service. It also limits the possibility of spreading the load over other instances.
Enable Always On to ensure Web Jobs run reliably.	A web app can time out after 20 minutes of inactivity. Only requests to the actual web app reset the timer. If your app runs continuously or if you schedule Timer trigger Web Jobs, enable Always On. Only available in the Basic, Standard, and Premium pricing tiers.
Access the on-prem database using private connections like Azure VPN or Express Route.	Access the on-premises database through a private connection so that the connection is secure and predictable.
Set up backup and restore.	Backup and restore lets you manually create or schedule app backups. You can retain backups for an indefinite amount of time.

ASE Recommendation	Description
Understand IP Address deprecation impact.	Floating addresses aren't guaranteed to remain on the resource. It's essential to check for deprecated IP addresses.
Deploy in highly available configuration across Availability Zones.	Ensures applications can continue to operate even if there's a datacenter-level failure. This provides excellent redundancy without requiring multiple deployments in different Azure regions.
Configure ASE Network correctly.	A common ASE pitfall occurs when ASE is deployed into a subnet with an IP address space that is too small to support future expansion. In such cases, ASE can be left unable to scale without redeploying the entire environment into a larger subnet. We highly recommend that adequate IP addresses be used to support either the maximum number of workers or the largest number considered workloads will need. A single ASE cluster can scale to `201` instances, which would require a `/24` subnet.
Configure **Upgrade preference** if multiple environments are used.	If lower environments are used for staging or testing, consider configuring these environments to receive updates sooner than the production environment. This will help to identify any conflicts or problems with an update, and provides a window to mitigate issues before they reach the production environment. If multiple load balanced (zonal) production deployments are used, *Upgrade preference* can be used to protect the broader environment against issues from platform upgrades.
Scale out the ASE cluster.	Scaling ASE instances vertically or horizontally takes `30` to `60` minutes as new private instances need to be provisioned. We highly recommend investing in up-front planning for scaling during spikes in load or transient failure scenarios.
Use **Deployment slots** for resilient code deployments.	*Deployment slots* allow for code to be deployed to instances that are *warmed-up* before serving production traffic. For more information, reference Testing in production with Azure App Service.
Avoid unnecessary worker restarts.	Many events can lead App Service workers to restart, such as content deployment, App Settings changes, and VNet integration configuration changes. A best practice is to make changes in a deployment slot other than the slot currently configured to accept production traffic. After workers are recycled and warmed up, a *swap* can be performed without unnecessary down time.

ASE Recommendation	Description
`Run From Package` to avoid deployment conflicts	`Run from Package` provides several advantages: - Eliminates file lock conflicts between deployment and runtime. - Ensures only fully deployed apps are running at any time. - May reduce *cold-start* times, particularly for JavaScript functions with large `npm` package trees.
Use Basic or higher plans with two or more worker instances for high availability.	Azure App Service provides many configuration options that aren't enabled by default.
Evaluate the use of TCP and SNAT ports.	TCP connections are used for all outbound connections; but, SNAT ports are used when making outbound connections to public IP addresses. SNAT port exhaustion is a common failure scenario that can be predicted by load testing while monitoring ports using Azure Diagnostics. For more information, reference TCP and SNAT ports.
Enable **Health check** to identify non-responsive workers.	Any health check is better than none at all. The logic behind endpoint tests should assess all critical downstream dependencies to ensure overall health. As a best practice, we highly recommend tracking application health and cache status in real time as this removes unnecessary delays before action can be taken.
Enable **Autoscale** to ensure adequate resources are available to service requests.	The default limit of App Service workers is `30`. If the App Service routinely uses `15` or more instances, consider opening a support ticket to increase the maximum number of workers to `2x` the instance count required to serve normal peak load.
Enable `Local_Cache` to reduce dependencies on cluster file servers.	Enabling local cache is always appropriate because it can lead to slower worker startup times. When coupled with **Deployment slots**, it can improve resiliency by removing dependencies on file servers and also reduces storage-related recycle events. Don't use local cache with a single worker instance or when shared storage is required.
Enable **Diagnostic Logging** to provide insight into application behavior.	*Diagnostic logging* provides the ability to ingest rich application and platform-level logs through Log Analytics, Azure Storage, or a third-party tool using Event Hub.

ASE Recommendation	Description
Enable **Application Insights alerts** to make you aware of fault conditions.	Application performance monitoring with Application Insights provides deep analyses into application performance. For Windows Plans, a *codeless deployment* approach is possible to quickly get a performance analysis without changing any code.
Review **Azure App Service diagnostics** to ensure common problems are addressed.	It's a good practice to regularly review service-related diagnostics and recommendations, and take action as appropriate.
Evaluate **per-app scaling** for high density hosting on Azure App Service.	Per-app scaling can be enabled at the App Service plan level to allow for scaling an app independently from the App Service plan that hosts it. This way, an App Service plan can be scaled to `10` instances, but an app can be set to use only five. Apps are allocated within the available App Service plan using a best effort approach for even distribution across instances. While an even distribution isn't guaranteed, the platform will make sure that two instances of the same app won't be hosted on the same App Service plan instance.

TCP and SNAT ports

If a load test results in SNAT errors, it's necessary to either scale across more or larger workers, or implement coding practices to help preserve and reuse SNAT ports, such as connection pooling and the lazy loading of resources. We don't recommend exceeding `100` simultaneous outbound connections to a public IP address per worker, and to avoid communicating with downstream services through public IP addresses when a private address (Private Endpoint) or Service Endpoint through vNet Integration could be used. TCP port exhaustion happens when the sum of connection from a given worker exceeds the capacity. The number of available TCP ports depend on the size of the worker.

The following table lists the current limits:

TCP ports	Small (B1, S1, P1, I1)	Medium (B2, S2, P2, I2)	Large (B3, S3, P3, I3)
TCP ports	1920	3968	8064

Applications with many longstanding connections require ports to be left open for long periods of time, which can lead to TCP Connection exhaustion. TCP Connection limits are fixed based on instance size, so it's necessary to scale up to a larger worker size to increase the allotment of TCP connections, or implement code level mitigations to govern connection usage. Similar to SNAT port exhaustion, you can use Azure Diagnostics to identify a problem exists with TCP port limits.

Source artifacts

To identify App Service Plans with only one instance, use the following query:

SQL

```sql
Resources
| where type == "microsoft.web/serverfarms" and properties.computeMode ==
`Dedicated`
| where sku.capacity == 1
```

Learn more

The Ultimate Guide to Running Healthy Apps in the Cloud

Next step

Azure App Service and cost optimization

Azure App Service and cost optimization

Article • 03/06/2023

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. This service adds the power of Microsoft Azure to your application, such as:

- Security
- Load balancing
- Autoscaling
- Automated management

To explore how to optimize costs for Azure App Service in your workload, reference key features in Why use App Service?

The following sections include a checklist and recommended configuration options specific to Azure App Service.

Checklist

Have you configured Azure App Service while considering cost optimization?

- ✓ Ensure the ASE subnet is appropriately sized.
- ✓ Consider cost savings by using the App Service Premium v3 plan over the Premium v2 plan.
- ✓ Always use a scale-out and scale-in rule combination.
- ✓ Understand the behavior of multiple scaling rules in a profile.
- ✓ Consider Basic or Free tier for non-production usage.

Configuration recommendations

Explore the following table of recommendations to optimize your App Service configuration for service cost:

ASE Recommendation	Description

ASE Recommendation	Description
Ensure the ASE subnet is appropriately sized.	The size of the subnet used to host an ASE directly affects maximum scale. An ASE with no App Service plans will use `12` to `13` addresses before you create an app. It's recommended that you deploy ASEs into a `/24` subnet. The maximum number of nodes in an ASE is `100`. During a scale-up event, the new machines are provisioned and placed into the subnet before the applications are migrated to the new machines, and the old machines are removed. The subnet must allow for at least `200` machines to handle the maximum deployment size, which requires a `/24` subnet. If you plan for insufficient capacity, scale-out operations will be limited.
Use App Service Premium v3 plan over the Premium v2 plan	The App Service Premium (v3) Plan has a `20%` discount versus comparable Pv2 configurations. Reserved Instance commitment (1Y, 3Y, Dev/Test) discounts are available for App Services running in the Premium v3 plan. Consider adding an Azure savings plan for compute and get significant savings on compute services with a one or three year commitment.
Use a scale-out and scale-in rule combination	If you use only one part of the combination, autoscale will only take action in a single direction (scale out, or in) until it reaches the maximum, or minimum instance counts defined in the profile. This scaling behavior isn't optimal, ideally you want your resource to scale up at times of high usage to ensure availability. Similarly, at times of low usage, you want your resource to scale down, so you can realize cost savings.
Understand the behavior of multiple scaling rules in a profile.	There are cases where you may have to set multiple rules in a profile. On scale-out, autoscale runs if `any` rule is met. On scale-in, autoscale requires `all` rules to be met.
Consider Basic or Free tier for non-production usage.	For non-prod App Service plans, consider scaling to Basic or Free Tier and scale up, as needed, and scale down when not in use – for example, during a Load Test exercise or based on the capabilities provided (such as custom domain, SSL, and more).

Next step

Azure App Service and operational excellence

Azure App Service and operational excellence

Article • 11/30/2022

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. This service adds the power of Microsoft Azure to your application, such as:

- Security
- Load balancing
- Autoscaling
- Automated management

To explore how Azure App Service can benefit the operational excellence of your application workload, reference key features in Why use App Service?

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure App Service.

Design considerations

Microsoft guarantees that Azure App Service will be available `99.95%` of the time. However, no SLA is provided using either the Free or Shared tiers.

For more information, reference the SLA for App Service.

Checklist

Have you configured Azure App Service while considering operational excellence?

- ✓ Create a deployment plan because redeploying the app service can reset the scaled units.
- ✓ Review the App Service Advisor recommendations.
- ✓ Ensure you configure the App Service Environments (ASE) Network correctly.
- ✓ Consider configuring Upgrade Preference if you're using multiple environments.
- ✓ Plan for scaling out the ASE cluster.
- ✓ Use Deployment Slots for resilient code deployments.
- ✓ Avoid unnecessary worker restarts when deploying application code or configuration.
- ✓ Use Run From Package to avoid deployment conflicts.

- ✓ Use Basic or higher plans with two or more worker instances for high availability.
- ✓ Evaluate the use of TCP and SNAT ports to avoid outbound connection errors.
- ✓ Enable Health check to identify non-responsive workers.
- ✓ Enable Autoscale to ensure adequate resources are available to service requests.
- ✓ Enable Local Cache to reduce dependencies on cluster file servers.
- ✓ Enable Diagnostic Logging to provide insight into application behavior.
- ✓ Enable Application Insights Alerts to signal fault conditions.
- ✓ Review Azure App Service diagnostics to ensure common problems are addressed.
- ✓ Evaluate per-app scaling for high density hosting on Azure App Service.

ASE Recommendation	Description
Create a deployment plan because redeploying the app service can reset the scaled units.	Automatic scaling rules apply during operation of the environment, but redeploying the app service may cause the plan to reset to the default number of units. Customers should be aware of this behavior and plan for it during deployments. Deploy only during off-peak times or deploy maximum units with automatic scaling enabled to scale in and out to prevent website performance implications.
Review the App Service Advisor recommendations.	App Service Advisor gives you real-time recommendations in the portal on resource exhaustion and conditions related to CPU, memory, and connections.
Ensure you configure the **App Service Environments (ASE) Network** correctly.	One common ASE pitfall occurs when ASE is deployed into a subnet with an IP address space that is too small to support future expansion. In such cases, ASE can be left unable to scale without redeploying the entire environment into a larger subnet. We highly recommended that adequate IP addresses be used to support either the maximum number of workers or the largest number considered workloads will need. A single ASE cluster can scale to `201` instance, which would require a `/24` subnet.
Configure **Upgrade preference** if you're using multiple environments.	If lower environments are used for staging or testing, consider configuring these environments to receive updates sooner than the production environment. This will help to identify any conflicts or problems with an update, and provides a window to mitigate issues before they reach the production environment. If multiple load balanced (zonal) production deployments are used, *Upgrade preference* can be used to protect the broader environment against issues from platform upgrades.
Scale out the ASE cluster.	Scaling ASE instances vertically or horizontally takes `30` to `60` minutes as new private instances need to be provisioned. We highly recommend investing in up-front planning for scaling during spikes in load or transient failure scenarios.

ASE Recommendation	Description
Use **Deployment slots** for resilient code deployments.	*Deployment slots* allow for code to be deployed to instances that are *warmed-up* before serving production traffic. For more information, reference Testing in production with Azure App Service.
Avoid unnecessary worker restarts.	Many events can lead App Service workers to restart, such as content deployment, App Settings changes, and VNet integration configuration changes. A best practice is to make changes in a deployment slot other than the slot currently configured to accept production traffic. After workers are recycled and warmed up, a *swap* can be performed without unnecessary down time.
`Run From Package` to avoid deployment conflicts	`Run from Package` provides several advantages: - Eliminates file lock conflicts between deployment and runtime. - Ensures only fully deployed apps are running at any time. - May reduce *cold-start* times, particularly for JavaScript functions with large `npm` package trees.
Use Basic or higher plans with two or more worker instances for high availability.	Azure App Service provides many configuration options that aren't enabled by default.
Evaluate the use of TCP and SNAT ports.	TCP connections are used for all outbound connections, but SNAT ports are used when making outbound connections to public IP addresses. SNAT port exhaustion is a common failure scenario that can be predicted by load testing while monitoring ports using Azure Diagnostics. For more information, reference TCP and SNAT ports.
Enable **Health check** to identify non-responsive workers.	Any health check is better than none at all. The logic behind endpoint tests should assess all critical downstream dependencies to ensure overall health. As a best practice, we highly recommend tracking application health and cache status in real time as this removes unnecessary delays before action can be taken.
Enable **Autoscale** to ensure adequate resources are available to service requests.	The default limit of App Service workers is `30`. If the App Service routinely uses `15` or more instances, consider opening a support ticket to increase the maximum number of workers to `2x` the instance count required to serve normal peak load.

ASE Recommendation	Description
Enable `Local_Cache` to reduce dependencies on cluster file servers.	Enabling local cache is always appropriate because it can lead to slower worker startup times. When coupled with **Deployment slots**, it can improve resiliency by removing dependencies on file servers and also reduces storage-related recycle events. Don't use local cache with a single worker instance or when shared storage is required.
Enable **Diagnostic Logging** to provide insight into application behavior.	*Diagnostic logging* provides the ability to ingest rich application and platform-level logs through Log Analytics, Azure Storage, or a third-party tool using Event Hub.
Enable **Application Insights alerts** to make you aware of fault conditions.	Application performance monitoring with Application Insights provides deep analyses into application performance. For Windows Plans, a *codeless deployment* approach is possible to quickly get a performance analysis without changing any code.
Review **Azure App Service diagnostics** to ensure common problems are addressed.	It's a good practice to regularly review service-related diagnostics and recommendations, and take action as appropriate.
Evaluate **per-app scaling** for high density hosting on Azure App Service.	Per-app scaling can be enabled at the App Service plan level to allow for scaling an app independently from the App Service plan that hosts it. This way, an App Service plan can be scaled to `10` instances, but an app can be set to use only five. Apps are allocated to available App Service plan using a best effort approach for an even distribution across instances. While an even distribution isn't guaranteed, the platform will make sure that two instances of the same app won't be hosted on the same App Service plan instance.

TCP and SNAT ports

If a load test results in SNAT errors, it's necessary to either scale across more or larger workers, or implement coding practices to help preserve and reuse SNAT ports, such as connection pooling and the lazy loading of resources. We don't recommend exceeding `100` simultaneous outbound connections to a public IP address per worker, and to avoid communicating with downstream services through public IP addresses when a private address (Private Endpoint) or Service Endpoint through vNet Integration could be used. TCP port exhaustion happens when the sum of connection from a given worker exceeds the capacity. The number of available TCP ports depend on the size of the worker.

The following table lists the current limits:

TCP ports	Small (B1, S1, P1, I1)	Medium (B2, S2, P2, I2)	Large (B3, S3, P3, I3)
TCP ports	1920	3968	8064

Applications with many longstanding connections require ports to be left open for long periods of time, which can lead to TCP Connection exhaustion. TCP Connection limits are fixed based on instance size, so it's necessary to scale up to a larger worker size to increase the allotment of TCP connections, or implement code level mitigations to govern connection usage. Similar to SNAT port exhaustion, you can use Azure Diagnostics to identify if a problem exists with TCP port limits.

Source artifacts

To identify App Service plans with only one instance, use the following query:

```SQL
Resources
| where type == "microsoft.web/serverfarms" and properties.computeMode ==
`Dedicated`
| where sku.capacity == 1
```

Learn more

The Ultimate Guide to Running Healthy Apps in the Cloud

Next step

Azure Batch and reliability

Azure Batch and reliability

Article • 11/30/2022

Azure Batch allows you to run large-scale parallel and high-performance computing (HPC) batch jobs efficiently in Azure.

Use Azure Batch to:

- Create and manage a pool of compute nodes (virtual machines).
- Install applications you want to run.
- Schedule jobs to run on the compute nodes.

The following sections include a design and configuration checklist, recommended design, and configuration options specific to Azure Batch.

Design and configuration checklist

Have you designed your workload and configured Azure Batch with resiliency in mind?

- ✓ Keep application binaries and reference data up to date in all regions.
- ✓ Use fewer jobs and more tasks.
- ✓ Use multiple Batch accounts in various regions to allow your application to continue running, if an Azure Batch account in one region becomes unavailable.
- ✓ Build durable tasks.
- ✓ Pre-create all required services in each region, such as the Batch account and storage account.
- ✓ Make sure the appropriate quotas are set on all subscriptions ahead of time, so you can allocate the required number of cores using the Batch account.

Design and configuration recommendations

Explore the following table of recommendations to optimize your workload design and Azure Batch configuration for service reliability:

Recommendation	Description
Keep application binaries and reference data up to date in all regions.	Staying up to date will ensure the region can be brought online quickly without waiting for file upload and deployment.

Recommendation	Description
Use fewer jobs and more tasks.	Using a job to run a single task is inefficient. For example, it's more efficient to use a single job containing `1000` tasks rather than creating `100` jobs that contain `10` tasks each. Running `1000` jobs, each with a single task, would be the least efficient, slowest, and most expensive approach.
Use multiple Batch accounts in various regions to allow your application to continue running, if an Azure Batch account in one region becomes unavailable.	It's crucial to have multiple accounts for a highly available application.
Build durable tasks.	Tasks should be designed to withstand failure and accommodate retry, especially for long running tasks. Ensure tasks generate the same, single result even if they're run more than once. One way to achieve the same result is to make your tasks *goal seeking*. Another way is to make sure your tasks are *idempotent* (tasks will have the same outcome no matter how many times they're run).
Pre-create all required services in each region, such as the Batch account and storage account.	There's often no charge for creating accounts and charges accrue only when you use the account, or when you store data.

Next step

Azure Batch and operational excellence

Azure Batch and operational excellence

Article • 11/30/2022

Azure Batch allows you to run large-scale parallel and high-performance computing (HPC) batch jobs efficiently in Azure.

Use Azure Batch to:

- Create and manage a pool of compute nodes (virtual machines).
- Install applications you want to run.
- Schedule jobs to run on the compute nodes.

The following sections include a design and configuration checklist, recommended design, and configuration options specific to Azure Batch.

Design and configuration checklist

Have you designed your workload and configured Azure Batch with operational excellence in mind?

- ✓ Keep application binaries and reference data up to date in all regions.
- ✓ Use fewer jobs and more tasks.
- ✓ Pre-create all required services in each region, such as the Batch account and storage account.
- ✓ Make sure the appropriate quotas are set on all subscriptions ahead of time, so you can allocate the required number of cores using the Batch account.

Design and configuration recommendations

Explore the following table of recommendations to optimize your workload design and Azure Batch configuration for operational excellence:

Recommendation	Description
Keep application binaries and reference data up to date in all regions.	Staying up to date will ensure the region can be brought online quickly without waiting for file upload and deployment.

Recommendation	Description
Use fewer jobs and more tasks.	Using a job to run a single task is inefficient. For example, it's more efficient to use a single job containing `1000` tasks rather than creating `100` jobs that contain `10` tasks each. Running `1000` jobs, each with a single task, would be the least efficient, slowest, and most expensive approach.
Pre-create all required services in each region, such as the Batch account and storage account.	There's often no charge for creating accounts and charges accrue only when you use the account, or when you store data.

Next step

Azure Batch and performance efficiency

Azure Batch and performance efficiency

Article • 11/30/2022

Azure Batch allows you to run large-scale parallel and high-performance computing (HPC) batch jobs efficiently in Azure.

Use Azure Batch to:

- Create and manage a pool of compute nodes (virtual machines).
- Install applications you want to run.
- Schedule jobs to run on the compute nodes.

The following sections include a design checklist and recommended design options specific to Azure Batch.

Design checklist

Have you designed your workload and configured Azure Batch with performance efficiency in mind?

✓ Use fewer jobs and more tasks.

Design and configuration recommendations

Consider the following recommendation to optimize your workload design and Azure Batch configuration for performance efficiency:

Recommendation	Description
Use fewer jobs and more tasks.	Using a job to run a single task is inefficient. For example, it's more efficient to use a single job containing `1000` tasks rather than creating `100` jobs that contain `10` tasks each. Running `1000` jobs, each with a single task, would be the least efficient, slowest, and most expensive approach.

Next step

AKS and reliability

Azure Well-Architected Framework review - Azure Kubernetes Service (AKS)

Article • 04/17/2023

This article provides architectural best practices for Azure Kubernetes Service (AKS). The guidance is based on the five pillars of architecture excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

We assume that you understand system design principles, have working knowledge of Azure Kubernetes Service, and are well versed with its features. For more information, see Azure Kubernetes Service.

Prerequisites

Understanding the Well-Architected Framework pillars can help produce a high-quality, stable, and efficient cloud architecture. We recommend that you review your workload by using the Azure Well-Architected Framework Review assessment.

For context, consider reviewing a reference architecture that reflects these considerations in its design. We recommend that you start with the baseline architecture for an Azure Kubernetes Service (AKS) cluster and Microservices architecture on Azure Kubernetes Service. Also review the AKS landing zone accelerator, which provides an architectural approach and reference implementation to prepare landing zone subscriptions for a scalable Azure Kubernetes Service (AKS) cluster.

Reliability

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Use the following information to minimize failed instances.

When discussing reliability with Azure Kubernetes Service, it's important to distinguish between *cluster reliability* and *workload reliability*. Cluster reliability is a shared responsibility between the cluster admin and their resource provider, while workload reliability is the domain of a developer. Azure Kubernetes Service has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

Design checklist

- ✓ **Cluster architecture:** For critical workloads, use availability zones for your AKS clusters.
- ✓ **Cluster architecture:** Plan the IP address space to ensure your cluster can reliably scale, including handling of failover traffic in multi-cluster topologies.
- ✓ **Cluster architecture:** Enable Container insights to monitor your cluster and configure alerts for reliability-impacting events.
- ✓ **Workload architecture:** Ensure workloads are built to support horizontal scaling and report application readiness and health.
- ✓ **Cluster and workload architectures:** Ensure your workload is running on user node pools and chose the right size SKU. At a minimum, include two nodes for user node pools and three nodes for the system node pool.
- ✓ **Cluster architecture:** Use the AKS Uptime SLA to meet availability targets for production workloads.

AKS configuration recommendations

Explore the following table of recommendations to optimize your AKS configuration for Reliability.

Recommendation	Benefit
Cluster and workload architectures: Control pod scheduling using node selectors and affinity.	Allows the Kubernetes scheduler to logically isolate workloads by hardware in the node. Unlike tolerations, pods without a matching node selector can be scheduled on labeled nodes, which allows unused resources on the nodes to consume, but gives priority to pods that define the matching node selector. Use node affinity for more flexibility, which allows you to define what happens if the pod can't be matched with a node.
Cluster architecture: Ensure proper selection of network plugin based on network requirements and cluster sizing.	Azure CNI is required for specific scenarios, for example, Windows-based node pools, specific networking requirements and Kubernetes Network Policies. Reference Kubenet versus Azure CNI for more information.

Recommendation	Benefit
Cluster and workload architectures: Use the AKS Uptime SLA for production grade clusters.	The AKS Uptime SLA guarantees: - `99.95%` availability of the Kubernetes API server endpoint for AKS Clusters that use Azure Availability Zones, or - `99.9%` availability for AKS Clusters that don't use Azure Availability Zones.
Cluster and workload architectures: Configure monitoring of cluster with Container insights.	Container insights help monitor the health and performance of controllers, nodes, and containers that are available in Kubernetes through the Metrics API. Integration with Prometheus enables collection of application and workload metrics.
Cluster architecture: Use availability zones to maximize resilience within an Azure region by distributing AKS agent nodes across physically separate data centers.	By spreading node pools across multiple zones, nodes in one node pool will continue running even if another zone has gone down. If colocality requirements exist, either a regular VMSS-based AKS deployment into a single zone or proximity placement groups can be used to minimize internode latency.
Cluster architecture: Adopt a multiregion strategy by deploying AKS clusters deployed across different Azure regions to maximize availability and provide business continuity.	Internet facing workloads should leverage Azure Front Door or Azure Traffic Manager to route traffic globally across AKS clusters.
Cluster and workload architectures: Define Pod resource requests and limits in application deployment manifests, and enforce with Azure Policy.	Container CPU and memory resource limits are necessary to prevent resource exhaustion in your Kubernetes cluster.
Cluster and workload architectures: Keep the System node pool isolated from application workloads.	System node pools require a VM SKU of at least 2 vCPUs and 4 GB memory, but 4 vCPU or more is recommended. Reference System and user node pools for detailed requirements.
Cluster and workload architectures: Separate applications to dedicated node pools based on specific requirements.	Applications may share the same configuration and need GPU-enabled VMs, CPU or memory optimized VMs, or the ability to scale-to-zero. Avoid large number of node pools to reduce extra management overhead.

Recommendation	Benefit
Cluster architecture: Use a NAT gateway for clusters that run workloads that make many concurrent outbound connections.	To avoid reliability issues with Azure Load Balancer limitations with high concurrent outbound traffic, us a NAT Gateway instead to support reliable egress traffic at scale.

For more suggestions, see Principles of the reliability pillar.

Azure Policy

Azure Kubernetes Service offers a wide variety of built-in Azure Policies that apply to both the Azure resource like typical Azure Policies and, using the Azure Policy add-on for Kubernetes, also within the cluster. There are a numerous number of policies, and key policies related to this pillar are summarized here. For a more detailed view, see built-in policy definitions for Kubernetes.

Cluster and workload architecture

- Clusters have readiness or liveness health probes configured for your pod spec.

In addition to the built-in Azure Policy definitions, custom policies can be created for both the AKS resource and for the Azure Policy add-on for Kubernetes. This allows you to add additional reliability constraints you'd like to enforce in your cluster and workload architecture.

Security

Security is one of the most important aspects of any architecture. To explore how AKS can bolster the security of your application workload, we recommend you review the Security design principles. If your Azure Kubernetes Service cluster needs to be designed to run a sensitive workload that meets the regulatory requirements of the Payment Card Industry Data Security Standard (PCI-DSS 3.2.1), review AKS regulated cluster for PCI-DSS 3.2.1.

To learn about DoD Impact Level 5 (IL5) support and requirements with AKS, review Azure Government IL5 isolation requirements.

When discussing security with Azure Kubernetes Service, it's important to distinguish between *cluster security* and *workload security*. Cluster security is a shared responsibility between the cluster admin and their resource provider, while workload security is the domain of a developer. Azure Kubernetes Service has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

Design checklist

- ✓ **Cluster architecture:** Use Managed Identities to avoid managing and rotating service principles.
- ✓ **Cluster architecture:** Use Kubernetes role-based access control (RBAC) with Azure AD for least privilege access and minimize granting administrator privileges to protect configuration, and secrets access.
- ✓ **Cluster architecture:** Use Microsoft Defender for containers with Azure Sentinel to detect and quickly respond to threats across your cluster and workloads running on them.
- ✓ **Cluster architecture:** Deploy a private AKS cluster to ensure cluster management traffic to your API server remains on your private network. Or use the API server allow list for non-private clusters.
- ✓ **Workload architecture:** Use a Web Application Firewall to secure HTTP(S) traffic.
- ✓ **Workload architecture:** Ensure your CI/CID pipeline is hardened with container-aware scanning.

Recommendations

Explore the following table of recommendations to optimize your AKS configuration for security.

Recommendation	Benefit
Cluster architecture: Use Azure Active Directory integration.	Using Azure AD centralizes the identity management component. Any change in user account or group status is automatically updated in access to the AKS cluster. The developers and application owners of your Kubernetes cluster need access to different resources.
Cluster architecture: Authenticate with Azure Active Directory (Azure AD) to Azure Container Registry.	AKS and Azure AD enables authentication with Azure Container Registry without the use of `imagePullSecrets` secrets. Review Authenticate with Azure Container Registry from Azure Kubernetes Service for more information.

Recommendation	Benefit
Cluster architecture: Secure network traffic to your API server with private AKS cluster.	By default, network traffic between your node pools and the API server travels the Microsoft backbone network; by using a private cluster, you can ensure network traffic to your API server remains on the private network only.
Cluster architecture: For non-private AKS clusters, use API server authorized IP ranges.	When using public clusters, you can still limit the traffic that can reach your clusters API server by using the authorized IP range feature. Include sources like the public IPs of your deployment build agents, operations management, and node pools' egress point (such as Azure Firewall).
Cluster architecture: Protect the API server with Azure Active Directory RBAC.	Securing access to the Kubernetes API Server is one of the most important things you can do to secure your cluster. Integrate Kubernetes role-based access control (RBAC) with Azure AD to control access to the API server. Disable local accounts to enforce all cluster access using Azure AD-based identities.
Cluster architecture: Use Azure network policies or Calico.	Secure and control network traffic between pods in a cluster.
Cluster architecture: Secure clusters and pods with Azure Policy.	Azure Policy can help to apply at-scale enforcement and safeguards on your clusters in a centralized, consistent manner. It can also control what functions pods are granted and if anything is running against company policy.
Cluster architecture: Secure container access to resources.	Limit access to actions that containers can perform. Provide the least number of permissions, and avoid the use of root or privileged escalation.
Workload architecture: Use a Web Application Firewall to secure HTTP(S) traffic.	To scan incoming traffic for potential attacks, use a web application firewall such as Azure Web Application Firewall (WAF) on Azure Application Gateway or Azure Front Door.
Cluster architecture: Control cluster egress traffic.	Ensure your cluster's outbound traffic is passing through a network security point such as Azure Firewall or an HTTP proxy.
Cluster architecture: Use the open-source Azure AD Workload Identity and Secrets Store CSI Driver with Azure Key Vault.	Protect and rotate secrets, certificates, and connection strings in Azure Key Vault with strong encryption. Provides an access audit log, and keeps core secrets out of the deployment pipeline.
Cluster architecture: Use Microsoft Defender for Containers.	Monitor and maintain the security of your clusters, containers, and their applications.

For more suggestions, see Principles of the security pillar.

Azure Advisor helps ensure and improve Azure Kubernetes service. It makes recommendations on a subset of the items listed in the policy section below, such as clusters without RBAC configured, missing Microsoft Defender configuration, unrestricted network access to the API Server. Likewise, it makes workload recommendations for some of the pod security initiative items. Review the recommendations.

Policy definitions

Azure Policy offers various built-in policy definitions that apply to both the Azure resource and AKS like standard policy definitions, and using the Azure Policy add-on for Kubernetes, also within the cluster. Many of the Azure resource policies come in both *Audit/Deny*, but also in a *Deploy If Not Exists* variant.

There are a numerous number of policies, and key policies related to this pillar are summarized here. For a more detailed view, see built-in policy definitions for Kubernetes.

Cluster architecture

- Microsoft Defender for Cloud-based policies
- Authentication mode and configuration policies (Azure AD, RBAC, disable local authentication)
- API Server network access policies, including private cluster

Cluster and workload architecture

- Kubernetes cluster pod security initiatives Linux-based workloads
- Include pod and container capability policies such as AppArmor, sysctl, security caps, SELinux, seccomp, privileged containers, automount cluster API credentials
- Mount, volume drivers, and filesystem policies
- Pod/Container networking policies, such as host network, port, allowed external IPs, HTTPs, and internal load balancers

Azure Kubernetes Service deployments often also use Azure Container Registry for Helm charts and container images. Azure Container Registry also supports a wide variety of Azure policies that spans network restrictions, access control, and Microsoft Defender for Cloud, which complements a secure AKS architecture.

In addition to the built-in policies, custom policies can be created for both the AKS resource and for the Azure Policy add-on for Kubernetes. This allows you to add additional security constraints you'd like to enforce in your cluster and workload architecture.

For more suggestions, see AKS security concepts and evaluate our security hardening recommendations based on the CIS Kubernetes benchmark.

Cost optimization

Cost optimization is about understanding your different configuration options and recommended best practices to reduce unnecessary expenses and improve operational efficiencies. Before you follow the guidance in this article, we recommend you review the following resources:

- Cost optimization design principles.
- How pricing and cost management work in Azure Kubernetes Service (AKS) compared to Amazon Elastic Kubernetes Service (Amazon EKS).
- If you are running AKS on-premises or at the edge, Azure Hybrid Benefit can also be used to further reduce costs when running containerized applications in those scenarios.

When discussing cost optimization with Azure Kubernetes Service, it's important to distinguish between *cost of cluster resources* and *cost of workload resources*. Cluster resources are a shared responsibility between the cluster admin and their resource provider, while workload resources are the domain of a developer. Azure Kubernetes Service has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations**, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

For cluster cost optimization, go to the Azure pricing calculator and select **Azure Kubernetes Service** from the available products. You can test different configuration and payment plans in the calculator.

Design checklist

- ✓ **Cluster architecture:** Use appropriate VM SKU per node pool and reserved instances where long-term capacity is expected.
- ✓ **Cluster and workload architectures:** Use appropriate managed disk tier and size.
- ✓ **Cluster architecture:** Review performance metrics, starting with CPU, memory, storage, and network, to identify cost optimization opportunities by cluster, nodes,

and namespace.

- ✓ **Cluster architecture:** Use cluster autoscaler to scale in when workloads are less active.

Recommendations

Explore the following table of recommendations to optimize your AKS configuration for cost.

Recommendation	Benefit
Cluster and workload architectures: Align SKU selection and managed disk size with workload requirements.	Matching your selection to your workload demands ensures you don't pay for unneeded resources.
Cluster architecture: Select the right virtual machine instance type.	Selecting the right virtual machine instance type is critical as it directly impacts the cost of running applications on AKS. Choosing a high-performance instance without proper utilization can lead to wasteful spending, while choosing a powerful instance can lead to performance issues and increased downtime. To determine the right virtual machine instance type, consider workload characteristics, resource requirements, and availability needs.
Cluster architecture: Select virtual machines based on the Arm architecture.	AKS supports creating ARM64 Ubuntu agent nodes, as well as a of mix Intel and ARM architecture nodes within a cluster that can bring better performance at a lower cost.
Cluster architecture: Select the appropriate region.	Due to many factors, cost of resources varies per region in Azure. Evaluate the cost, latency, and compliance requirements to ensure you are running your workload cost-effectively and it doesn't affect your end-users or create extra networking charges.
Workload architecture: Maintain small and optimized images.	Streamlining your images helps reduce costs since new nodes need to download these images. Build images in a way that allows the container start as soon as possible to help avoid user request failures or timeouts while the application is starting up, potentially leading to overprovisioning.

Recommendation	Benefit
Cluster architecture: Enable cluster autoscaler to automatically reduce the number of agent nodes in response to excess resource capacity.	Automatically scale down the number of nodes in your AKS cluster lets you run an efficient cluster when demand is low, scale up when demand returns.
Workload architecture: Use the Horizontal Pod Autoscaler.	Adjust the number of pods in a deployment depending on CPU utilization or other select metrics, which support cluster scale-in operations.
Workload architecture: Use Vertical Pod Autoscaler (preview).	Rightsize your pods and dynamically set requests and limits based on historic usage.
Workload architecture: Use Kubernetes Event Driven Autoscaling (KEDA).	Scale based on the number of events being processed. Choose from a rich catalogue of 50+ KEDA scalers.
Cluster and workload architectures: Adopt a cloud financial discipline and cultural practice to drive ownership of cloud usage.	The foundation of enabling cost optimization is the spread of a cost saving cluster. A financial operations approach (FinOps) is often used to help organizations reduce cloud costs. It is a practice involving collaboration between finance, operations, and engineering teams to drive alignment on cost saving goals and bring transparency to cloud costs.
Cluster architecture: Sign up for Azure Reservations or Azure Savings Plan.	If you properly planned for capacity, your workload is predictable and exists for an extended period of time, sign up for an Azure Reservation or a savings plan to further reduce your resource costs.
Cluster architecture: Configure monitoring of cluster with Container insights.	Container insights help provides actionable insights into your clusters idle and unallocated resources. Container insights also supports collecting Prometheus metrics and integrates with Azure Managed Grafana to get a holistic view of your application and infrastructure.

For more suggestions, see Principles of the cost optimization pillar.

Policy definitions

While there are no built-in policies that are related to cost optimization, custom policies can be created for both the AKS resource and for the Azure Policy add-on for Kubernetes. This allows you to add additional cost optimization constraints you'd like to enforce in your cluster and workload architecture.

Cloud efficiency

Making workloads more sustainable and cloud efficient, requires combining efforts around **cost optimization**, **reducing carbon emissions**, and **optimizing energy consumption**. Optimizing the application's cost is the initial step in making workloads more sustainable.

Learn how to build sustainable and efficient AKS workloads, in Sustainable software engineering principles in Azure Kubernetes Service (AKS).

Operational excellence

Monitoring and diagnostics are crucial. Not only can you measure performance statistics, but also use metrics troubleshoot and remediate issues quickly. We recommend you review the Operational excellence design principles and the Day-2 operations guide.

When discussing operational excellence with Azure Kubernetes Service, it's important to distinguish between *cluster operational excellence* and *workload operational excellence*. Cluster operations are a shared responsibility between the cluster admin and their resource provider, while workload operations are the domain of a developer. Azure Kubernetes Service has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

Design checklist

- ✔ **Cluster architecture:** Use a template-based deployment using Bicep, Terraform, or others. Make sure that all deployments are repeatable, traceable, and stored in a source code repo.
- ✔ **Cluster architecture:** Build an automated process to ensure your clusters are bootstrapped with the necessary cluster-wide configurations and deployments. This

is often performed using GitOps.

- ✓ **Workload architecture:** Use a repeatable and automated deployment processes for your workload within your software development lifecycle.
- ✓ **Cluster architecture:** Enable diagnostics settings to ensure control plane or core API server interactions are logged.
- ✓ **Cluster and workload architectures:** Enable Container insights to collect metrics, logs, and diagnostics to monitor the availability and performance of the cluster and workloads running on it.
- ✓ **Workload architecture:** The workload should be designed to emit telemetry that can be collected, which should also include liveliness and readiness statuses.
- ✓ **Cluster and workload architectures:** Use chaos engineering practices that target Kubernetes to identify application or platform reliability issues.
- ✓ **Workload architecture:** Optimize your workload to operate and deploy efficiently in a container.
- ✓ **Cluster and workload architectures:** Enforce cluster and workload governance using Azure Policy.

Recommendations

Explore the following table of recommendations to optimize your AKS configuration for operations.

Recommendation	Benefit
Cluster and workload architectures: Review AKS best practices documentation.	To build and run applications successfully in AKS, there are key considerations to understand and implement. These areas include multi-tenancy and scheduler features, cluster, and pod security, or business continuity and disaster recovery.
Cluster and workload architectures: Review Azure Chaos Studio.	Azure Chaos Studio can help simulate faults and trigger disaster recovery situations.
Cluster and workload architectures: Configure monitoring of cluster with Container insights.	Container insights help monitor the performance of containers by collecting memory and processor metrics from controllers, nodes, and containers that are available in Kubernetes through the Metrics API and container logs.
Workload architecture: Monitor application performance with Azure Monitor.	Configure Application Insights for code-based monitoring of applications running in an AKS cluster.

Recommendation	Benefit
Workload architecture: Configure scraping of Prometheus metrics with Container insights.	Container insights, which are part of Azure Monitor, provide a seamless onboarding experience to collect Prometheus metrics. Reference Configure scraping of Prometheus metrics for more information.
Cluster architecture: Adopt a multiregion strategy by deploying AKS clusters deployed across different Azure regions to maximize availability and provide business continuity.	Internet facing workloads should leverage Azure Front Door or Azure Traffic Manager to route traffic globally across AKS clusters.
Cluster architecture: Operationalize clusters and pods configuration standards with Azure Policy.	Azure Policy can help to apply at-scale enforcement and safeguards on your clusters in a centralized, consistent manner. It can also control what functions pods are granted and if anything is running against company policy.
Workload architecture: Use platform capabilities in your release engineering process.	Kubernetes and ingress controllers support many advanced deployment patterns for inclusion in your release engineering process. Consider patterns like blue-greem deployments or canary releases.
Cluster and workload architectures: For mission-critical workloads, use stamp-level blue/green deployments.	Automate your mission-critical design areas, including deployment and testing.

For more suggestions, see Principles of the operational excellence pillar.

Azure Advisor also makes recommendations on a subset of the items listed in the policy section below, such unsupported AKS versions and unconfigured diagnostic settings. Likewise, it makes workload recommendations around the use of the default namespace.

Policy definitions

Azure Policy offers various built-in policy definitions that apply to both the Azure resource and AKS like standard policy definitions, and using the Azure Policy add-on for Kubernetes, also within the cluster. Many of the Azure resource policies come in both *Audit/Deny*, but also in a *Deploy If Not Exists* variant.

There are a numerous number of policies, and key policies related to this pillar are summarized here. For a more detailed view, see built-in policy definitions for Kubernetes.

Cluster architecture

- Azure Policy add-on for Kubernetes
- GitOps configuration policies
- Diagnostics settings policies
- AKS version restrictions
- Prevent command invoke

Cluster and workload architecture

- Namespace deployment restrictions

In addition to the built-in policies, custom policies can be created for both the AKS resource and for the Azure Policy add-on for Kubernetes. This allows you to add additional security constraints you'd like to enforce in your cluster and workload architecture.

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. We recommend you review the Performance efficiency principles.

When discussing performance with Azure Kubernetes Service, it's important to distinguish between *cluster performance* and *workload performance*. Cluster performance is a shared responsibility between the cluster admin and their resource provider, while workload performance is the domain of a developer. Azure Kubernetes Service has considerations and recommendations for both of these roles.

In the **design checklist** and **list of recommendations** below, call-outs are made to indicate whether each choice is applicable to cluster architecture, workload architecture, or both.

Design checklist

As you make design choices for Azure Kubernetes Service, review the Performance efficiency principles.

✓ **Cluster and workload architectures:** Perform and iterate on a detailed capacity plan exercise that includes SKU, autoscale settings, IP addressing, and failover considerations.

- ✓ **Cluster architecture:** Enable cluster autoscaler to automatically adjust the number of agent nodes in response workload demands.
- ✓ **Cluster architecture:** Use the Horizontal pod autoscaler to adjust the number of pods in a deployment depending on CPU utilization or other select metrics.
- ✓ **Cluster and workload architectures:** Perform ongoing load testing activities that exercise both the pod and cluster autoscaler.
- ✓ **Cluster and workload architectures:** Separate workloads into different node pools allowing independent scalling.

Recommendations

Explore the following table of recommendations to optimize your Azure Kubernetes Service configuration for performance.

Recommendation	Benefit
Cluster and workload architectures: Develop a detailed capacity plan and continually review and revise.	After formalizing your capacity plan, it should be frequently updated by continuously observing the resource utilization of the cluster.
Cluster architecture: Enable cluster autoscaler to automatically adjust the number of agent nodes in response to resource constraints.	The ability to automatically scale up or down the number of nodes in your AKS cluster lets you run an efficient, cost-effective cluster.
Cluster and workload architectures: Separate workloads into different node pools and consider scaling user node pools.	Unlike System node pools that always require running nodes, user node pools allow you to scale up or down.
Workload architecture: Use AKS advanced scheduler features.	Helps control balancing of resources for workloads that require them.
Workload architecture: Use meaningful workload scaling metrics.	Not all scale decisions can be derived from CPU or memory metrics. Often scale considerations will come from more complex or even external data points. Use KEDA to build a meaningful auto scale ruleset based on signals that are specific to your workload.

For more suggestions, see Principles of the performance efficiency pillar.

Policy definitions

Azure Policy offers various built-in policy definitions that apply to both the Azure resource and AKS like standard policy definitions, and using the Azure Policy add-on for Kubernetes, also within the cluster. Many of the Azure resource policies come in both *Audit/Deny*, but also in a *Deploy If Not Exists* variant.

There are a numerous number of policies, and key policies related to this pillar are summarized here. For a more detailed view, see built-in policy definitions for Kubernetes.

Cluster and workload architecture

- CPU and memory resource limits

In addition to the built-in policies, custom policies can be created for both the AKS resource and for the Azure Policy add-on for Kubernetes. This allows you to add additional security constraints you'd like to enforce in your cluster and workload architecture.

Additional resources

Azure Architecture Center guidance

- AKS baseline architecture
- Advanced AKS microservices architecture
- AKS cluster for a PCI-DSS workload
- AKS baseline for multiregion clusters

Cloud Adoption Framework guidance

- AKS Landing Zone Accelerator

Next steps

- Deploy an Azure Kubernetes Service (AKS) cluster using the Azure CLI Quickstart: Deploy an Azure Kubernetes Service (AKS) cluster using the Azure CLI

Azure Functions and security

Article • 11/30/2022

Azure Functions is a cloud service available on-demand that provides all the continually updated infrastructure and resources needed to run your applications. Functions allow you to write less code, maintain less infrastructure, and save on costs. Instead of worrying about deploying and maintaining servers, the cloud infrastructure provides all the up-to-date resources needed to keep your applications securely running.

For more information related to network security, reference Securing Azure Functions.

The following sections include a design consideration checklist and recommendations specific to Azure Functions, and security.

Design consideration checklist

Have you designed your workload and configured Azure Functions with security in mind?

- ✓ Evaluate if Azure Functions requires HTTP trigger.
- ✓ Treat Azure Functions code just like any other code.
- ✓ Use guidance available on Securing Azure Functions.
- ✓ Consider using Azure Functions Proxy to act as a facade.

Design consideration recommendations

The following table reflects design consideration recommendations and descriptions related to Azure Functions:

Azure Functions Design Recommendations	Description

Azure Functions Design Recommendations	Description
Evaluate if Azure Functions requires HTTP trigger.	Azure Functions supports multiple specific triggers and bindings. These include Azure Blob storage, Azure Cosmos DB, Azure Service Bus, and many more. If HTTP trigger is needed, then consider protecting that HTTP endpoint like any other web application. Common protection measures include keeping HTTP endpoint internal to specific Azure virtual networks by using Private endpoint connections or service endpoints. Consider using guidance available on Azure Functions networking options for more information. If Functions HTTP endpoint will be exposed to internet, then it's recommended to secure the endpoint behind a web application firewall (WAF).
Treat Azure Functions code just like any other code.	Subject Azure Functions code to code scanning tools that are integrated with CI/CD pipeline.
Use guidance available on Securing Azure Functions.	This guidance addresses key security concerns such as operations, deployment, and network security.
Consider using Azure Functions Proxy to act as a facade.	Functions Proxy can inspect and modify incoming requests and responses.

Next step

Azure Service Fabric and reliability

Azure Well-Architected Framework review - Virtual Machines

Article • 04/27/2023

Virtual Machines is an on-demand, scalable computing resource that gives you the flexibility of virtualization without having to buy and maintain physical hardware to run it.

In this article, you learn architectural best practices for Azure Virtual Machines. The guidance is based on the five pillars of architectural excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

Prerequisites

- Understanding the Well-Architected Framework pillars can help produce a high quality, stable, and efficient cloud architecture. We recommend that you review your workload using the Microsoft Azure Well-Architected Review assessment.

- Use a reference architecture to review the considerations based on the guidance provided in this article. We recommend, you start with Run a Linux VM on Azure.

Reliability

As you make design choices for virtual machines, review the design principles for adding reliability to the architecture.

Design checklist

- ✓ Review the SLAs for virtual machines .
- ✓ VMs should be deployed in a scale set using the Flexible orchestration mode.
- ✓ Deployed VMs across Availability Zones .
- ✓ Install applications on data disks.
- ✓ Use maintenance control.

Recommendations

Explore the following table of recommendations to optimize your Virtual Machine configuration for service reliability:

Recommendation	Benefit
Review SLAs for virtual machines.	When defining test availability and recovery targets, make sure you have a good understanding of the SLAs offered for VMs.
Deploy using Flexible scale sets.	Even single instance VMs should be deployed into a scale set using the Flexible orchestration mode to future-proof your application for scaling and availability. Flexible orchestration offers high availability guarantees (up to 1000 VMs) by spreading VMs across fault domains in a region or within an Availability Zone.
Deploy across availability zones	Azure availability zones are physically separate locations within each Azure region that are tolerant to local failures.
Install applications on data disks.	Having your data on a separate disk from your OS disk makes it easier to recover from failures and to migrate workloads.
Use maintenance control	Control when VM maintenance occurs to manage the timing of system restarts.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the Azure Advisor recommendations.

Security

This article provides an overview of the core Azure security features that can be used with virtual machines.

As you make design choices for virtual machines, review the security principles and Security best practices for adding security to the architecture.

Design checklist

As you make design choices for your virtual machine deployment, review the design principles for security.

- ✓ Review the Linux security baseline
- ✓ Review the Windows security baseline
- ✓ Manage authentication and access control.
- ✓ Protect against malware

- ✓ Managed updates
- ✓ Encryption

Recommendations

Explore the following table of recommendations to optimize your virtual machine configuration for security.

Recommendation	Benefit
Consider using Azure Bastion	Authentication and access control using Azure Bastion provides secure and seamless RDP/SSH connectivity to your virtual machines directly from the Azure portal over TLS
Protect against malware	Install antimalware protection to help identify and remove viruses.
Manage updates	Use a solution like Azure Automation to manage operating system updates.
Monitor for security	To monitor the security posture of your Windows and Linux VMs, use Microsoft Defender for Cloud.
Use encryption	Use Azure Disk Encryption to protect your data.

For more suggestions, see Principles of the security pillar.

Azure Advisor helps you ensure and improve security. Review the recommendations.

Policy definitions

- `Deploy default Microsoft IaaSAntimalware extension for Windows Server` - This policy deploys a Microsoft IaaSAntimalware extension with a default configuration when a VM is not configured with the antimalware extension.
- `Microsoft IaaSAntimalware extension should be deployed on Windows servers` - This policy audits any Windows server VM without Microsoft IaaSAntimalware extension deployed.
- `Only approved VM extensions should be installed` - This policy governs the virtual machine extensions that are not approved.
- `Managed disks should be double encrypted with both platform-managed and customer-managed keys` - High security sensitive customers who are concerned of the risk associated with any particular encryption algorithm, implementation, or key being compromised can opt for additional layer of encryption using a different encryption algorithm/mode at the infrastructure layer using platform managed

encryption keys. The disk encryption sets are required to use double encryption. Learn more at https://aka.ms/disks-doubleEncryption.

- `Managed disks should use a specific set of disk encryption sets for the customer-managed key encryption` - Requiring a specific set of disk encryption sets to be used with managed disks give you control over the keys used for encryption at rest. You are able to select the allowed encrypted sets and all others are rejected when attached to a disk. Learn more at https://aka.ms/disks-cmk.
- `Microsoft Antimalware for Azure should be configured to automatically update protection signatures` - This policy audits any Windows virtual machine not configured with automatic update of Microsoft Antimalware protection signatures.
- `OS and data disks should be encrypted with a customer-managed key` - Use customer-managed keys to manage the encryption at rest of the contents of your managed disks. By default, the data is encrypted at rest with platform-managed keys, but customer-managed keys are commonly required to meet regulatory compliance standards. Customer-managed keys enable the data to be encrypted with an Azure Key Vault key created and owned by you. You have full control and responsibility for the key lifecycle, including rotation and management. Learn more at https://aka.ms/disks-cmk.
- `Virtual machines and virtual machine scale sets should have encryption at host enabled` - Use encryption at host to get end-to-end encryption for your virtual machine and virtual machine scale set data. Encryption at host enables encryption at rest for your temporary disk and OS/data disk caches. Temporary and ephemeral OS disks are encrypted with platform-managed keys when encryption at host is enabled. OS/data disk caches are encrypted at rest with either customer-managed or platform-managed key, depending on the encryption type selected on the disk. Learn more at https://aka.ms/vm-hbe.
- `Require automatic OS image patching on Virtual Machine Scale Sets` - This policy enforces enabling automatic OS image patching on Virtual Machine Scale Sets to always keep virtual Machines secure by safely applying latest security patches every month.

All built-in policy definitions related to Azure Virtual Machines are listed in Azure Policy built-in definitions for Azure Virtual Machines.

Cost optimization

To optimize costs, review the design principles.

To estimate costs related to virtual machines, use these tools.

- Identify the best VM for your workloads with the virtual machines selector. For more information, see Linux and Windows pricing.
- Use this pricing calculator to configure and estimate the costs of your Azure VMs.

Design considerations

- ✓ Shut down VM instances which aren't in use.
- ✓ Use Spot VMs when appropriate.
- ✓ Choose the right VM size for your workload.
- ✓ Use Zone to Zone disaster recovery for virtual machines.
- ✓ Prepay for reserved instances or an Azure savings plan for compute for significant savings.
- ✓ Use hybrid benefit licensing

Recommendations

Explore the following table of recommendations to optimize your Virtual Machine configuration for service cost:

Recommendation	Benefit
Stop VMs during off-hours	Configuring start and stop times will shut down instances that aren't in use. The feature is suitable as a low-cost automation option.
Use Spot VMs when appropriate.	Spot VMs are ideal for workloads that can be interrupted, such as highly parallel batch processing jobs. These VMs take advantage of the surplus capacity in Azure at a lower cost. They're also well suited for experimenting, developing, and testing large-scale solutions. Check out our Azure Virtual Machine Spot Eviction guide to learn how to create a reliable interruptible workload in Azure.
Right-size your VMs	Identify the best VM for your workloads with the virtual machines selector. See Windows and Linux pricing.
Prepay for added cost savings	Purchasing reserved instances is a way to reduce Azure costs for workloads with stable usage. Make sure you manage usage. If usage is too low, then you're paying for resources that aren't used. Keep reserved instances simple and keep management overhead low to prevent increasing cost. With an Azure savings plan for compute, you save money across select compute services globally. Commit to spend a fixed hourly amount for 1 or 3 years and unlock lower prices until you reach your hourly commitment.

| Use existing licensing through the hybrid benefit licensing program | Hybrid benefit licensing is available for both Linux and Windows|

Azure Advisor helps you ensure and improve cost optimization. Review the recommendations.

Policy definitions

- Consider setting an `Allowed virtual machine SKU` policy to limit the sizes that can be used.

All built-in policy definitions related to Azure Virtual Machines are listed in Azure Policy built-in definitions for Azure Virtual Machines.

Operational excellence

To ensure operational excellence, review the design principles.

Design checklist

- ✓ Monitor and measure health.
- ✓ Automate tasks like provisioning and updating.
- ✓ Build a robust testing environment.
- ✓ Right size your VMs.
- ✓ Manage your quota.

Recommendations

Recommendation	Benefit
Monitor and measure health	In a production environment, it's important to monitor the health, and performance of your VMs.
Automate tasks	Building automation reduces deviations from your plans and reduces that time it takes to manage your workload.
Build a robust testing environment	Ideally, an organization will have multiple environments in which to test deployments. These test environments should be similar enough to production that deployment and run time issues are detected before deployment to production.
Right-size your VMs	Choose the right VM family for your workload.
Manage your quota	Plan what level of quota will be required and review that level regularly as the workload evolves and grows and request changes early

For more suggestions, see Principles of the operational excellence pillar.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the recommendations.

Policy definitions

- Consider setting an `Allowed virtual machine SKU` policy

All built-in policy definitions related to Azure Virtual Machines are listed in Azure Policy built-in definitions for Azure Virtual Machines.

Performance efficiency

Performance efficiency is matching the resources that are available to an application with the demand that it's receiving. Performance efficiency includes scaling resources, identifying and optimizing potential bottlenecks, and optimizing your application code for peak performance.

As you make design choices for your virtual machine deployment, review Microsoft Azure Well-Architected Framework - Performance efficiency for performance and efficiency.

Design checklist

- ✓ Reduce latency by deploying VMs closer together in proximity placement groups
- ✓ Convert disks from standard HDD to premium SSD
- ✓ Enable Accelerated Networking to improve network performance and latency
- ✓ Autoscale your Flexible scale sets.

Recommendations

Explore the following table of recommendations to optimize your virtual machine deployment configuration for performance and efficiency.

Recommendation	Benefit
Reduce latency	Consider deploying VMs in Creating and using proximity placement groups using PowerShell.

Recommendation	Benefit
Convert disks from standard HDD to premium SSD	Azure premium SSDs deliver high-performance and low-latency disk support for virtual machines (VMs) with input/output (IO)-intensive workloads.
Consider accelerated networking	Accelerated networking enables single root I/O virtualization (SR-IOV) to a VM, greatly improving its networking performance.
Use autoscaling	Automatically increase or decrease the number of VM instances that run your application with autoscaling.

For more suggestions, see Principles of the performance efficiency pillar.

Azure Advisor helps you ensure and improve performance. Review the recommendations.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your Virtual Machines.

- Reliability
- Cost Optimization
- Performance
- Operational excellence

Additional resources

Here are other resources to help you query for unhealthy instances.

Cost analysis

Planned versus actual spending can be managed through Azure Cost Management + Billing. There are several options for grouping resources by billing unit.

Next steps

Use the recommendations as you provision virtual machines for your solution.

- Learn module: Introduction to Azure virtual machines

- Review the Virtual Machine recommendations provided by Azure Advisor.
- Review the built-in definitions provided by Azure Policy that apply to Virtual Machines. All built-in policy definitions related to Azure Virtual Machines are listed in Azure Policy built-in definitions for Azure Virtual Machines.

Azure Cache for Redis and reliability

Article • 11/30/2022

Azure Cache for Redis provides an in-memory data store based on the Redis (Remote Dictionary Server) software. It's a secure data cache and messaging broker that provides high throughput and low-latency access to data for applications.

Key concepts and best practices that support reliability include:

- High availability
- Failover and patching
- Connection resilience

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Cache for Redis.

Design considerations

The Azure Cache for Redis Service Level Agreements (SLA) covers only Standard and Premium tier caches. Basic tier isn't covered.

Redis is an in-memory cache for key value pairs and has High Availability (HA), by default, except for Basic tier. There are three tiers for Azure Cache for Redis:

- *Basic*: *Not recommended for production workloads*. Basic tier is ideal for:
  - Single node
  - Multiple sizes
  - Development
  - Test
  - Non-critical workloads

- *Standard*: A replicated cache in a two-node primary and secondary configuration managed by Microsoft, with a high availability SLA.

- *Premium*: Includes all standard-tier features and includes the following other features:
  - Faster hardware and performance compared to Basic or Standard tier.
  - Larger cache size, up to `120GB`.
  - Data persistence, which includes Redis Database File (RDB) and Append Only File (AOF).
  - VNET support.
  - Clustering

- Geo-Replication: A secondary cache is in another region and replicates data from the primary for disaster recovery. To failover to the secondary, the caches need to be unlinked manually and then the secondary is available for writes. The application writing to Redis needs to be updated with the secondary's cache connection string.
- Availability Zones: Deploy the cache and replicas across availability zones.

> **Note**
>
> By default, each deployment will have one replica per shard. Persistence, clustering, and geo-replication are all disabled at this time with deployments that have more than one replica. Your nodes will be distributed evenly across all zones. You should have a replica count `>=` number of zones.

- Import and export.

Microsoft guarantees at least `99.9%` of the time that customers will have connectivity between the Cache Endpoints and Microsoft's Internet gateway.

Checklist

Have you configured Azure Cache for Redis with resiliency in mind?

- ✓ Schedule updates.
- ✓ Monitor the cache and set alerts.
- ✓ Deploy the cache within a VNET.
- ✓ Evaluate a partitioning strategy within Redis cache.
- ✓ Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.
- ✓ Implement retry policies in the context of your Azure Redis Cache.
- ✓ Use one static or singleton implementation of the connection multiplexer to Redis and follow the best practices guide.
- ✓ Review How to administer Azure Cache for Redis.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Cache for Redis configuration for service reliability:

Recommendation	Description

Recommendation	Description
Schedule updates.	Schedule the days and times that Redis Server updates will be applied to the cache, which doesn't include Azure updates, or updates to the VM operating system.
Monitor the cache and set alerts.	Set alerts for exceptions, high CPU, high memory usage, server load, and evicted keys for insights about when to scale the cache. If the cache needs to be scaled, understanding when to scale is important because it will increase CPU during the scaling event to migrate data.
Deploy the cache within a VNET.	Gives the customer more control over the traffic that can connect to the cache. Make sure that the subnet has sufficient address space available to deploy the cache nodes and shards (cluster).
Evaluate a partitioning strategy within Redis cache.	Partitioning a Redis data store involves splitting the data across instances of the Redis server. Each instance makes up a single partition. Azure Redis Cache abstracts the Redis services behind a facade and doesn't expose them directly. The simplest way to implement partitioning is to create multiple Azure Redis Cache instances and spread the data across them. You can associate each data item with an identifier (a partition key) that specifies which cache stores the data item. The client application logic can then use this identifier to route requests to the appropriate partition. This scheme is simple, but if the partitioning scheme changes (for example, if extra Azure Redis Cache instances are created), client applications may need to be reconfigured.
Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.	*Data Persistence*: if the master and replica reboot, the data will be loaded automatically from the storage account. *Geo-Replication*: The secondary cache needs to be unlinked from the primary. The secondary will now become the primary and can receive *writes*.
Implement retry policies in the context of your Azure Redis Cache.	Most Azure services and client SDKs include a retry mechanism. These mechanisms differ because each service has different characteristics and requirements. Each retry mechanism is tuned to a specific service.
Review How to administer Azure Cache for Redis.	Understand how data loss can occur with cache reboots and how to test the application for resiliency.

Source artifacts

To identify Redis instances that aren't on the Premium tier, use the following query:

```sql
Resources
| where type == 'microsoft.cache/redis'
| where properties.sku.name != 'Premium'
```

Next step

[Azure Cache for Redis and operational excellence]

Azure Cache for Redis and operational excellence

Article • 11/30/2022

Azure Cache for Redis provides an in-memory data store based on the Redis (Remote Dictionary Server) software. It's a secure data cache and messaging broker that provides high throughput and low-latency access to data for applications.

Best practices that support operational excellence include:

- Server load management
- Memory management
- Performance testing

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Cache for Redis.

Design considerations

The Azure Cache for Redis Service Level Agreements (SLA) covers only Standard and Premium tier caches. Basic tier isn't covered.

Redis is an in-memory cache for key value pairs and has High Availability (HA), by default, except for Basic tier. There are three tiers for Azure Cache for Redis:

- *Basic*: *Not recommended for production workloads*. Basic tier is ideal for:
  - Single node
  - Multiple sizes
  - Development
  - Test
  - Non-critical workloads

- *Standard*: A replicated cache in a two-node primary and secondary configuration managed by Microsoft, with a high availability SLA.

- *Premium*: Includes all standard-tier features and includes the following other features:
  - Faster hardware and performance compared to Basic or Standard tier.
  - Larger cache size, up to `120GB`.
  - Data persistence, which includes Redis Database File (RDB) and Append Only File (AOF).

- VNET support.
- Clustering
- Geo-Replication: A secondary cache is in another region and replicates data from the primary for disaster recovery. To failover to the secondary, the caches need to be unlinked manually and then the secondary is available for writes. The application writing to Redis will need to be updated with the secondary's cache connection string.
- Availability Zones: Deploy the cache and replicas across availability zones.

> **Note**
>
> By default, each deployment will have one replica per shard. Persistence, clustering, and geo-replication are all disabled at this time with deployments that have more than one replica. Your nodes will be distributed evenly across all zones. You should have a replica count `>=` number of zones.

- Import and export.

Microsoft guarantees at least `99.9%` of the time that customers will have connectivity between the Cache Endpoints and Microsoft's Internet gateway.

Checklist

Have you configured Azure Cache for Redis with operational excellence in mind?

- ✓ Schedule updates.
- ✓ Monitor the cache and set alerts.
- ✓ Deploy the cache within a VNET.
- ✓ Use the correct caching type (local, in role, managed, redis) within your solution.
- ✓ Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.
- ✓ Use one static or singleton implementation of the connection multiplexer to Redis and follow the best practices guide.
- ✓ Review How to administer Azure Cache for Redis.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Cache for Redis configuration for operational excellence:

Recommendation	Description
Schedule updates.	Schedule the days and times that Redis Server updates will be applied to the cache, which doesn't include Azure updates, or updates to the VM operating system.
Monitor the cache and set alerts.	Set alerts for exceptions, high CPU, high memory usage, server load, and evicted keys for insights about when to scale the cache. If the cache needs to be scaled, understanding when to scale is important because it will increase CPU during the scaling event to migrate data.
Deploy the cache within a VNET.	Gives the customer more control over the traffic that can connect to the cache. Make sure that the subnet has sufficient address space available to deploy the cache nodes and shards (cluster).
Use the correct caching type (local, in role, managed, redis) within your solution.	Distributed applications typically implement either or both of the following strategies when caching data: - Using a private cache, where data is held locally on the machine that's running an instance of an application or service. - Using a shared cache, serving as a common source that can be accessed by multiple processes and machines. In both cases, caching can be performed client-side and server-side. Client-side caching is done by the process that provides the user interface for a system, such as a web browser or desktop application. Server-side caching is done by the process that provides the business services that are running remotely.
Configure **Data Persistence** to save a copy of the cache to Azure Storage or use Geo-Replication, depending on the business requirement.	*Data Persistence*: If the master and replica reboot, the data will be loaded automatically from the storage account. *Geo-Replication*: The secondary cache needs to be unlinked from the primary. The secondary will now become the primary and can receive *writes*.
Review How to administer Azure Cache for Redis.	Understand how data loss can occur with cache reboots and how to test the application for resiliency.

Source artifacts

To identify Redis instances that aren't on the Premium tier, use the following query:

SQL

```sql
Resources
| where type == 'microsoft.cache/redis'
| where properties.sku.name != 'Premium'
```

Next step

Azure Databricks and security

Azure Databricks and security

Article • 01/11/2023

Azure Databricks is a data analytics platform optimized for Azure cloud services. It offers three environments for developing data intensive applications:

- Databricks SQL
- Databricks Data Science and Engineering
- Databricks Machine Learning

To learn more about how Azure Databricks improves the security of big data analytics, reference Azure Databricks concepts.

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Databricks.

Design considerations

All users' notebooks and notebook results are encrypted at rest, by default. If other requirements are in place, consider using customer-managed keys for notebooks.

Checklist

Have you configured Azure Databricks with security in mind?

- ✓ Use Azure Active Directory credential passthrough to avoid the need for service principals when communicating with Azure Data Lake Storage.
- ✓ Isolate your workspaces, compute, and data from public access. Make sure that only the right people have access and only through secure channels.
- ✓ Ensure that the cloud workspaces for your analytics are only accessible by properly managed users.
- ✓ Implement Azure Private Link.
- ✓ Restrict and monitor your virtual machines.
- ✓ Use Dynamic IP access lists to allow admins to access workspaces only from their corporate networks.
- ✓ Use the VNet injection functionality to enable more secure scenarios.
- ✓ Use diagnostic logs to audit workspace access and permissions.
- ✓ Consider using the Secure cluster connectivity feature and hub/spoke architecture to prevent opening ports, and assigning public IP addresses on cluster nodes.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Databricks configuration for security:

Recommendation	Description
Ensure that the cloud workspaces for your analytics are only accessible by properly managed users.	Azure Active Directory can handle single sign-on for remote access. For extra security, reference Conditional Access.
Implement Azure Private Link.	Ensure all traffic between users of your platform, the notebooks, and the compute clusters that process queries are encrypted and transmitted over the cloud provider's network backbone, inaccessible to the outside world.
Restrict and monitor your virtual machines.	Clusters, which execute queries, should have SSH and network access restricted to prevent installation of arbitrary packages. Clusters should use only images that are periodically scanned for vulnerabilities.
Use the VNet injection functionality to enable more secure scenarios.	Such as: - Connecting to other Azure services using service endpoints. - Connecting to on-premises data sources, taking advantage of user-defined routes. - Connecting to a network virtual appliance to inspect all outbound traffic and take actions according to allow and deny rules. - Using custom DNS. - Deploying Azure Databricks clusters in existing virtual networks.
Use diagnostic logs to audit workspace access and permissions.	Use audit logs to see privileged activity in a workspace, cluster resizing, files, and folders shared on the cluster.

Source artifacts

Azure Databricks source artifacts include the Databricks blog: Best practices to secure an enterprise-scale data platform .

Next step

Azure Database for MySQL and cost optimization

Azure Database for MySQL and cost optimization

Article • 11/30/2022

Azure Database for MySQL is a relational database service in the Microsoft cloud based on the MySQL Community Edition. You can use either Single Server or Flexible Server to host a MySQL database in Azure. It's a fully managed database as a service offering that can handle mission-critical workloads with predictable performance and dynamic scalability.

For more information about how Azure Database for MySQL supports cost optimization for your workload, reference Server concepts, specifically, Stop/Start an Azure Database for MySQL.

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Database for MySQL.

Design considerations

Azure Database for MySQL includes the following design considerations:

- Take advantage of the scaling capabilities of Azure Database for MySQL to lower consumption cost whenever possible. To scale your database up and down, as needed, reference the following Microsoft Support article, which covers the automation process using runbooks: How to autoscale an Azure Database for MySQL/PostgreSQL instance with Azure run books and Python.
- Plan your Recovery Point Objective (RPO) according to your operation level requirement. There's no extra charge for backup storage for up to `100%` of your total provisioned server storage. Extra consumption of backup storage will be charged in `GB/month`.
- The cloud native design of the Single-Server service allows it to support `99.99%` of availability, eliminating the cost of passive *hot* standby.
- Consider using Flexible Server SKU for non-production workloads. Flexible servers provide better cost optimization controls with ability to stop and start your server. They provide a burstable compute tier that is ideal for workloads that don't need continuous full compute capacity.

Checklist

Have you configured Azure Database for MySQL with cost optimization in mind?

- ✓ Choose the appropriate server size for your workload.
- ✓ Consider Reserved Capacity for Azure Database for MySQL Single Server.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Database for MySQL configuration for cost optimization:

Recommendation	Description
Choose the appropriate server size for your workload.	Configuration options: Single Server and Flexible Server.
Consider Reserved Capacity for Azure Database for MySQL Single Server.	Compute costs associated with Azure Database For MySQL Single Server Reservation Discount. Once you've determined the total compute capacity and performance tier for Azure Database for MySQL in a region, this information can be used to reserve the capacity. The reservation can span one or three years. You can realize significant cost optimization with this commitment.

Azure Database for PostgreSQL and cost optimization

Azure Well-Architected Framework review for Azure Database for PostgreSQL

Article • 05/18/2023

Azure Database for PostgreSQL is a relational database service in Azure based on the PostgreSQL open source relational database. Azure Database for PostgreSQL has two deployment modes: Single Server and Flexible Server.

Single Server is a fully managed database service that provides high-availability, automated patching, automatic backups, dynamic scalability, and built-in performance monitoring. The Single Server deployment option is best suited for cloud native applications where you don't need to control the patching schedule or use custom PostgreSQL configuration settings.

Flexible Server is fully managed database service that provides more control over database management than the Single Server deployment option. You should use the Flexible Server deployment option when you need greater database control, cost optimization features, zone redundancy, and customizable maintenance windows.

This guide provides implementation guidance and recommendations for Azure Database for PostgreSQL. For Azure Cosmos DB for PostgreSQL, see Build scalable apps in Azure Cosmos DB for PostgreSQL.

Cost optimization

Design checklist

You should review the design principles to optimize the cost of your architecture.

- ✓ Pick the right tier and SKU
- ✓ Understand high availability mode
- ✓ Scale compute and storage tiers
- ✓ Consider reserved instances
- ✓ Use your provisioned storage
- ✓ Understand geo-redundancy costs
- ✓ Evaluate storage scale up decisions
- ✓ Deploy to same region as app

Recommendations

Azure Database for PostgreSQL recommendation	Description
Pick the right tier and SKU	Pick the pricing tier and compute SKUs that support the specific needs of your workload. Azure Advisor gives you recommendations to optimize and reduce your overall Azure spending. Recommendations include server right-sizing that you should follow.
Understand high availability mode	High availability makes a standby server always available within the same zone or region. Enabling high-availability doubles your cost.
Adjust compute and storage tiers	You should adjust the compute and storage tiers manually as needed to meet the requirements of the application over time.
Use Start/Stop feature	Flexible server has a Start/Stop feature that you can use to stop the server from running when you don't need it.
Consider reserved instances	Consider a one or three-year reservation to receive significant discounts on compute. Use these reservations for workloads with consistent compute usage for a year or more.
Use your provisioned storage	There's no extra charge for backup storage up to 100% of your total provisioned server storage.
Understand redundancy costs	Geo-redundant storage (GRS) costs twice as much as local redundant storage (LRS). GRS requires double the storage capacity of LRS.
Evaluate storage scale up decisions	You should evaluate your current and future storage needs before scaling up your storage. After you scale up storage, you can't scale down.
Deploy to same region as app	Deploy to the same region as the application(s) to minimize transfer costs. When you use virtual network integration, applications in a different virtual network don't have direct access to flexible servers. To grant them access, you need to configure virtual network peering. Virtual network peering has a nominal inbound and outbound data transfer costs.

Next step

Azure SQL Database

Azure Well-Architected Framework review - Azure SQL Database

Article • 06/24/2022

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most of the database management functions without user involvement. Management functions include upgrades, patches, backups, and monitoring.

The single database resource type creates a database in Azure SQL Database with its own set of resources and is managed via a logical server. You can choose between the DTU-based purchasing model or vCore-based purchasing model. You can create multiple databases in a single resource pool, with elastic pools.

The following sections include a design checklist and recommended design options specific to Azure SQL Database security. The guidance is based on the five pillars of architectural excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

Prerequisites

- Understanding the Well-Architected Framework pillars can help produce a high quality, stable, and efficient cloud architecture. Check out the Azure Well-Architected Framework overview page to review the five pillars of architectural excellence.

- Review the core concepts of Azure SQL Database and What's new in Azure SQL Database?.

Azure SQL Database and reliability

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most of the database management functions without user involvement. Management functions include:

- Upgrades

- Patches
- Backups
- Monitoring

This service allows you to create a highly available and high-performance data storage layer for your Azure applications and workloads. Azure SQL Database is always running on the latest stable version of the SQL Server database engine and patched OS with `99.99%` availability.

For more information about how Azure SQL Database promotes reliability and enables your business to continue operating during disruptions, reference Availability capabilities.

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure SQL Database and reliability.

Design considerations

Azure SQL Database includes the following design considerations:

- Azure SQL Database Business Critical tier configured with geo-replication has a guaranteed Recovery time objective (RTO) of `30` seconds for `100%` of deployed hours.

- Use *sharding* to distribute data and processes across many identically structured databases. Sharding provides an alternative to traditional scale-up approaches for cost and elasticity. Consider using sharding to partition the database horizontally. Sharding can provide fault isolation. For more information, reference Scaling out with Azure SQL Database.

- Azure SQL Database Business Critical or Premium tiers not configured for Zone Redundant Deployments, General Purpose, Standard, or Basic tiers, or Hyperscale tier with two or more replicas have an availability guarantee. For more information about the availability guarantee, reference SLA for Azure SQL Database.

- Provides built-in regional high availability and turnkey geo-replication to any Azure region. It includes intelligence to support self-driving features, such as:
  - Performance tuning
  - Threat monitoring
  - Vulnerability assessments
  - Fully automated patching and updating of the code base

- Define an application performance SLA and monitor it with alerts. Quickly detect when your application performance inadvertently degrades below an acceptable level, which is important to maintain high resiliency. Use the monitoring solution previously defined to set alerts on key query performance metrics so you can take action when the performance breaks the SLA. Go to Monitor Your Database and alerting tools for more information.

- Use geo-restore to recover from a service outage. You can restore a database on any SQL Database server or an instance database on any managed instance in any Azure region from the most recent geo-replicated backups. Geo-restore uses a geo-replicated backup as its source. You can request geo-restore even if the database or datacenter is inaccessible because of an outage. Geo-restore restores a database from a geo-redundant backup. For more information, reference Recover an Azure SQL database using automated database backups.

- Use the Business Critical tier configured with geo-replication, which has a guaranteed Recovery point objective (RPO) of `5` seconds for `100%` of deployed hours.

- PaaS capabilities built into Azure SQL Database enable you to focus on the domain-specific database administration and optimization activities that are critical for your business.

- Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from changes done inadvertently. For more information, read the Point-in-time restore (PITR) documentation.

- Business Critical or Premium tiers are configured as Zone Redundant Deployments which have an availability guarantee. For more information about the availability guarantee, reference SLA for Azure SQL Database .

Checklist

Have you configured Azure SQL Database with reliability in mind?

- ✓ Use Active Geo-Replication to create a readable secondary in a different region.
- ✓ Use Auto Failover Groups that can include one or multiple databases, typically used by the same application.
- ✓ Use a Zone-Redundant database.
- ✓ Monitor your Azure SQL Database in near-real time to detect reliability incidents.
- ✓ Implement Retry Logic.
- ✓ Back up your keys.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Database configuration for reliability:

Recommendation	Description
Use Active Geo-Replication to create a readable secondary in a different region.	If your primary database fails, perform a manual failover to the secondary database. Until you fail over, the secondary database remains read-only. Active geo-replication enables you to create readable replicas and manually failover to any replica if there is a datacenter outage or application upgrade. Up to four secondaries are supported in the same or different regions, and the secondaries can also be used for read-only access queries. The failover must be initiated manually by the application or the user. After failover, the new primary has a different connection end point.
Use Auto Failover Groups that can include one or multiple databases, typically used by the same application.	You can use the readable secondary databases to offload read-only query workloads. Because autofailover groups involve multiple databases, these databases must be configured on the primary server. Autofailover groups support replication of all databases in the group to only one secondary server or instance in a different region. Learn more about AutoFailover Groups and DR design.
Use a Zone-Redundant database.	By default, the cluster of nodes for the premium availability model is created in the same datacenter. With the introduction of Azure Availability Zones, SQL Database can place different replicas of the Business Critical database to different availability zones in the same region. To eliminate a single point of failure, the control ring is also duplicated across multiple zones as three gateway rings (GW). The routing to a specific gateway ring is controlled by Azure Traffic Manager (ATM). Because the zone redundant configuration in the Premium or Business Critical service tiers doesn't create extra database redundancy, you can enable it at no extra cost. Learn more about Zone-redundant databases.
Monitor your Azure SQL Database in near-real time to detect reliability incidents.	Use one of the available solutions to monitor SQL DB to detect potential reliability incidents early and make your databases more reliable. Choose a near real-time monitoring solution to quickly react to incidents. Reference Azure SQL Analytics for more information.
Implement Retry Logic.	Although Azure SQL Database is resilient when it concerns transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while working with SQL Database, make sure your code can retry the call. For more information, reference how to implement retry logic.

Recommendation	Description
Back up your keys.	If you're not using encryption keys in Azure Key Vault to protect your data, back up your keys.

Azure SQL Database and security

SQL Database provides a range of built-in security and compliance features to help your application meet various security and compliance requirements.

Design checklist

Have you designed your workload and configured Azure SQL Database with security in mind?

- ✓ Understand logical servers and how you can administer logins for multiple databases when appropriate.
- ✓ Enable Azure AD authentication with Azure SQL. Azure AD authentication enables simplified permission management and centralized identity management.
- ✓ Azure SQL logical servers should have an Azure Active Directory administrator provisioned.
- ✓ Verify contact information email address in your Azure Subscription for service administrator and co-administrators is reaching the correct parties inside your enterprise. You don't want to miss or ignore important security notifications from Azure!
- ✓ Review the Azure SQL Database connectivity architecture. Choose the `Redirect` or `Proxy` connection policy as appropriate.
- ✓ Review Azure SQL Database firewall rules.
- ✓ Use virtual network rules to control communication from particular subnets in virtual networks.
- ✓ If using the Azure Firewall, configure Azure Firewall application rules with SQL FQDNs.

Recommendations

Recommendation	Benefit

Recommendation	Benefit
Review the minimum TLS version.	Determine whether you have legacy applications that require older TLS or unencrypted connections. After you enforce a version of TLS, it's not possible to revert to the default. Review and configure the minimum TLS version for SQL Database connections via the Azure portal. If not, set the latest TLS version to the minimum.
Ledger	Consider designing database tables based on the Ledger to provide auditing, tamper-evidence, and trust of all data changes.
Always Encrypted	Consider designing application access based around Always Encrypted to protect sensitive data inside applications by delegating data access to encryption keys.
Private endpoints and private link	Private endpoint connections enforce secure communication by enabling private connectivity to Azure SQL Database. You can use a private endpoint to secure connections and deny public network access by default. Azure Private Link for Azure SQL Database is a type of private endpoint recommended for Azure SQL Database.
Automated vulnerability assessments	Monitor for vulnerability assessment scan results and recommendations for how to remediate database vulnerabilities.
Advanced Threat Protection	Detect anomalous activities indicating unusual and potentially harmful attempts to access or exploit databases with Advanced Threat Protection for Azure SQL Database. Advanced Threat Protection integrates its alerts with Microsoft Defender for Cloud .
Auditing	Track database events with Auditing for Azure SQL Database.
Managed identities	Consider configuring a user-assigned managed identity (UMI). Managed identities for Azure resources eliminate the need to manage credentials in code.
Azure AD-only authentication	Consider disabling SQL-based authentication and allowing only on Azure AD authentication.

Policy definitions

Review the Azure security baseline for Azure SQL Database and Azure Policy built-in definitions.

All built-in policy definitions related to Azure SQL are listed in Built-in policies.

Review Tutorial: Secure a database in Azure SQL Database.

Azure SQL Database and cost optimization

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most of the database management functions without user involvement. Management functions include:

- Upgrades
- Patches
- Backups
- Monitoring

This service allows you to create a highly available and high-performance data storage layer for your Azure applications and workloads. SQL Database includes built-in intelligence that helps you dramatically reduce the costs of running and managing databases through automatic performance monitoring and tuning.

For more information about how Azure SQL Database provides cost-saving features, reference Plan and manage costs for Azure SQL Database.

The following sections include a configuration checklist and recommended configuration options specific to Azure SQL Database and cost optimization.

Checklist

Have you configured Azure SQL Database with cost optimization in mind?

- ✓ Optimize queries.
- ✓ Evaluate resource usage.
- ✓ Fine-tune backup storage consumption.
- ✓ Evaluate Azure SQL Database serverless.
- ✓ Consider reserved capacity for Azure SQL Database.
- ✓ Consider elastic pools for managing and scaling multiple databases.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Database configuration for cost savings:

Recommendation	Description
Optimize queries.	Optimize the queries, tables, and databases using Query Performance Insights and Performance Recommendations to help reduce resource consumption, and arrive at appropriate configuration.

Recommendation	Description
Evaluate resource usage.	Evaluate the resource usage for all databases and determine if they've been sized and provisioned correctly. For non-production databases, consider scaling resources down as applicable. The DTUs or vCores for a database can be scaled on demand, for example, when running a load test or user acceptance test.
Fine-tune backup storage consumption	For vCore databases in Azure SQL Database, the storage consumed by each type of backup (full, differential, and log) is reported on the database monitoring pane as a separate metric. Backup storage consumption up to the maximum data size for a database is not charged. Excess backup storage consumption will depend on the workload and maximum size of the individual databases. For more information, see Backup storage consumption.
Evaluate Azure SQL Database Serverless.	Consider using Azure SQL Database serverless over the Provisioned Computing Tier. Serverless is a compute tier for single databases that automatically scales compute based on workload demand and bills for the amount of compute used per second. The serverless compute tier also automatically pauses databases during inactive periods when only storage is billed. It automatically resumes databases when activity returns. Azure SQL Database serverless isn't suited for all scenarios. If you have a database with unpredictable or bursty usage patterns interspersed with periods of low or idle usage, serverless is a solution that can help you optimize price-performance.
Consider reserved capacity for Azure SQL Database.	You can reduce compute costs associated with Azure SQL Database by using Reservation Discount. Once you've determined the total compute capacity and performance tier for Azure SQL databases in a region, you can use this information to reserve the capacity. The reservation can span one or three years. For more information, reference Save costs for resources with reserved capacity.
Elastic pools help you manage and scale multiple databases in Azure SQL Database	Azure SQL Database elastic pools are a simple, cost-effective solution for managing and scaling multiple databases that have varying and unpredictable usage demands. The databases in an elastic pool are on a single server and share a set number of resources at a set price. For more information, see Elastic pools for managing and scaling multiple databases.

For more information, see Plan and manage costs for Azure SQL Database.

Azure SQL Database and operational excellence

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most of the database management functions without user involvement. Management functions include:

- Upgrades
- Patches
- Backups
- Monitoring

This service allows you to create a highly available and high-performance data storage layer for your Azure applications and workloads. Azure SQL Database provides advanced monitoring and tuning capabilities backed by artificial intelligence to help you troubleshoot and maximize the performance of your databases and solutions.

For more information about how Azure SQL Database promotes operational excellence and enables your business to continue operating during disruptions, reference Monitoring and performance tuning in Azure SQL Database.

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure SQL Database, and operational excellence.

Design considerations

Azure SQL Database includes the following design considerations:

- Azure SQL Database Business Critical tier configured with geo-replication has a guaranteed Recovery time objective (RTO) of `30` seconds for `100%` of deployed hours.

- Use *sharding* to distribute data and processes across many identically structured databases. Sharding provides an alternative to traditional scale-up approaches for cost and elasticity. Consider using sharding to partition the database horizontally. Sharding can provide fault isolation. For more information, reference Scaling out with Azure SQL Database.

- Azure SQL Database Business Critical or Premium tiers not configured for Zone Redundant Deployments, General Purpose, Standard, or Basic tiers, or Hyperscale tier with two or more replicas have an availability guarantee. For more information, reference SLA for Azure SQL Database.

- Provides built-in regional high availability and turnkey geo-replication to any Azure region. It includes intelligence to support self-driving features, such as:
  - Performance tuning
  - Threat monitoring
  - Vulnerability assessments
  - Fully automated patching and updating of the code base

- Define an application performance SLA and monitor it with alerts. Quickly detect when your application performance inadvertently degrades below an acceptable level, which is important to maintain high resiliency. Use the monitoring solution previously defined to set alerts on key query performance metrics so you can take action when the performance breaks the SLA. Go to Monitor Your Database for more information.

- Use geo-restore to recover from a service outage. You can restore a database on any SQL Database server or an instance database on any managed instance in any Azure region from the most recent geo-replicated backups. Geo-restore uses a geo-replicated backup as its source. You can request geo-restore even if the database or datacenter is inaccessible because of an outage. Geo-restore restores a database from a geo-redundant backup. For more information, reference Recover an Azure SQL database using automated database backups.

- Use the Business Critical tier configured with geo-replication, which has a guaranteed Recovery point objective (RPO) of `5` seconds for `100%` of deployed hours.

- PaaS capabilities built into Azure SQL Database enable you to focus on the domain-specific database administration and optimization activities that are critical for your business.

- Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from changes done inadvertently. For more information, read the Point-in-time restore (PITR) documentation.

- Business Critical or Premium tiers are configured as Zone Redundant Deployments. For more information about the availability guarantee, reference SLA for Azure SQL Database .

Checklist

Have you configured Azure SQL Database with operational excellence in mind?

- ✓ Use Active Geo-Replication to create a readable secondary in a different region.
- ✓ Use Auto Failover Groups that can include one or multiple databases, typically used by the same application.
- ✓ Use a Zone-Redundant database.
- ✓ Monitor your Azure SQL Database in near-real time to detect reliability incidents.
- ✓ Implement retry logic.
- ✓ Back up your keys.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Database configuration for operational excellence:

Recommendation	Description
Use Active Geo-Replication to create a readable secondary in a different region.	If your primary database fails, perform a manual failover to the secondary database. Until you fail over, the secondary database remains read-only. Active geo-replication enables you to create readable replicas and manually failover to any replica if there is a datacenter outage or application upgrade. Up to four secondaries are supported in the same or different regions, and the secondaries can also be used for read-only access queries. The failover must be initiated manually by the application or the user. After failover, the new primary has a different connection end point.
Use Auto Failover Groups that can include one or multiple databases, typically used by the same application.	You can use the readable secondary databases to offload read-only query workloads. Because autofailover groups involve multiple databases, these databases must be configured on the primary server. Autofailover groups support replication of all databases in the group to only one secondary server or instance in a different region. Learn more about Auto-Failover Groups and DR design.
Use a Zone-Redundant database.	By default, the cluster of nodes for the premium availability model is created in the same datacenter. With the introduction of Azure Availability Zones, SQL Database can place different replicas of the Business Critical database to different availability zones in the same region. To eliminate a single point of failure, the control ring is also duplicated across multiple zones as three gateway rings (GW). The routing to a specific gateway ring is controlled by Azure Traffic Manager (ATM). Because the zone redundant configuration in the Premium or Business Critical service tiers doesn't create extra database redundancy, you can enable it at no extra cost. Learn more about Zone-redundant databases.
Monitor your Azure SQL Database in near-real time to detect reliability incidents.	Use one of the available solutions to monitor SQL DB to detect potential reliability incidents early and make your databases more reliable. Choose a near real-time monitoring solution to quickly react to incidents. Reference Azure SQL Analytics for more information.
Implement Retry Logic.	Although Azure SQL Database is resilient when it concerns transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while working with SQL Database, make sure your code can retry the call. For more information, reference how to implement retry logic and Configurable retry logic in SqlClient introduction.

Recommendation	Description
Back up your keys.	If you're not using encryption keys in Azure Key Vault to protect your data, back up your keys.

Azure SQL Database and performance efficiency

Azure SQL Database is a fully managed platform as a service (PaaS) database engine that handles most of the database management functions without user involvement. Management functions include:

- Upgrades
- Patches
- Backups
- Monitoring

The following sections include a design checklist and recommended design options specific to Azure SQL Database performance efficiency.

Design checklist

Have you designed your workload and configured Azure SQL Database with performance efficiency in mind?

- ✓ Review resource limits. For specific resource limits per pricing tier (also known as service objective) for single databases, refer to either DTU-based single database resource limits or vCore-based single database resource limits. For elastic pool resource limits, refer to either DTU-based elastic pool resource limits or vCore-based elastic pool resource limits.
- ✓ Choose the right deployment model for your workload, vCore or DTU. Compare the vCore and DTU-based purchasing models.
- ✓ Microsoft recommends the latest vCore database standard-series or premium-series hardware. Older Gen4 hardware has been retired.
- ✓ When using elastic pools, familiarize yourself with resource governance.
- ✓ Review the default max degree of parallelism (MAXDOP) and configure as needed based on a migrated or expected workload.
- ✓ Consider using read-only replicas of critical database to offload read-only query workloads.
- ✓ Review the Performance Center for SQL Server Database Engine and Azure SQL Database.

✓ Applications connecting to Azure SQL Database should use the latest connection providers, for example the latest OLE DB Driver or ODBC Driver.

Recommendations

Recommendation	Benefit
Diagnose and troubleshoot high CPU utilization.	Azure SQL Database provides built-in tools to identify the causes of high CPU usage and to optimize workload performance.
Understand blocking and deadlocking issues.	Blocking due to concurrency and terminated sessions due to deadlocks have different causes and outcomes.
Tune applications and databases for performance.	Tune your application and database to improve performance. Review best practices.
Review Azure portal utilization reporting and scale as appropriate.	After deployment, use built-in reporting in the Azure portal to regularly review peak and average database utilization and right-size up or down. You can easily scale single databases or elastic pools with no data loss and minimal downtime.
Review Performance Recommendations.	In the Intelligent Performance menu of the database page in the Azure portal, review and consider action on any of the Performance Recommendations and implement any index, schema, and parameterization issues.
Review Query Performance Insight.	Review Query Performance Insight for Azure SQL Database reports to identify top resource-consuming queries, long running queries, and more.
Configure Automatic tuning.	Provide peak performance and stable workloads through continuous performance tuning based on AI and machine learning. Consider using Azure Automation to configure email notifications for automatic tuning.
Evaluate potential use of in-memory database objects.	In-memory technologies enable you to improve performance of your application, and potentially reduce cost of your database. Consider designing some database objects in high-volume OLTP applications.
Leverage the Query Store.	Enabled by default in Azure SQL Database, the Query Store contains a wealth of query performance and resource consumption data, as well as advanced tuning features like Query Store hints and automatic plan correction. Review Query Store defaults in Azure SQL Database.

Recommendation	Benefit
Implement retry logic for transient errors.	Applications should include automatic transaction retry logic for transient errors including common connection errors. Leverage exponential retry interval logic.

Additional resources

For information about supported features, see Features and Resolving Transact-SQL differences during migration to SQL Database.

Migrating to Azure SQL Database? Review our Azure Database Migration Guides.

Watch episodes of Data Exposed covering Azure SQL topics and more.

Next steps

- Try Azure SQL Database free with Azure free account, then get started with single databases in Azure SQL Database.

Azure SQL Managed Instance and reliability

Article • 11/30/2022

Azure SQL Managed Instance is the intelligent, scalable cloud database service that combines the broadest SQL Server database engine compatibility with all the benefits of a fully managed and evergreen platform as a service.

The goal of the high availability architecture in SQL Managed Instance is to guarantee that your database is up and running without worrying about the impact of maintenance operations and outages. This solution is designed to:

- Ensure that committed data is never lost because of failures.
- Ensure that maintenance failures don't affect your workload.
- Ensure that the database won't be a single point of failure in your software architecture.

For more information about how Azure SQL Managed Instance supports application and workload resilience, reference the following articles:

- High availability for Azure SQL Managed Instance
- Use autofailover groups to enable transparent and coordinated geo-failover of multiple databases

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure SQL Managed Instance, and reliability.

Design considerations

Azure SQL Managed Instance includes the following design considerations:

- Define an application performance SLA and monitor it with alerts. Detecting quickly when your application performance inadvertently degrades below an acceptable level is important to maintain high resiliency. Use a monitoring solution to set alerts on key query performance metrics so you can take action when the performance breaks the SLA.
- Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from changes done inadvertently. For more information, read the Point-in-time-restore (PITR) documentation for managed instance.

- Use geo-restore to recover from a service outage. Geo-restore restores a database from a geo-redundant backup into a managed instance in a different region. For more information, reference Recover a database using Geo-restore documentation.
- Consider the time required for certain operations. Make sure you separate time to thoroughly test the amount of time required to scale up and down your existing managed instance, and to create a new managed instance. This timing practice ensures that you understand completely how time consuming operations will affect your RTO and RPO.

Checklist

Have you configured Azure SQL Managed Instance with reliability in mind?

- ✓ Use the Business Critical Tier.
- ✓ Configure a secondary instance and an Autofailover group to enable failover to another region.
- ✓ Implement Retry Logic.
- ✓ Monitor your SQL MI instance in near-real time to detect reliability incidents.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Managed Instance configuration for reliability:

Recommendation	Description
Use the Business Critical Tier.	This tier provides higher resiliency to failures and faster failover times because of the underlying HA architecture, among other benefits. For more information, reference SQL Managed Instance High availability.
Configure a secondary instance and an Autofailover group to enable failover to another region.	If an outage impacts one or more of the databases in the managed instance, you can manually or automatically failover all the databases inside the instance to a secondary region. For more information, read the Autofailover groups documentation for managed instance.
Implement Retry Logic.	Although Azure SQL MI is resilient to transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while working with SQL MI, make sure your code can retry the call. For more information, reference how to implement retry logic.

Recommendation	Description
Monitor your SQL MI instance in near-real time to detect reliability incidents.	Use one of the available solutions to monitor your SQL MI to detect potential reliability incidents early and make your databases more reliable. Choose a near real-time monitoring solution to quickly react to incidents. For more information, check out the Azure SQL Managed Instance monitoring options .

Next step

Azure SQL Managed Instance and operational excellence

Azure SQL Managed Instance and operational excellence

Article • 11/30/2022

Azure SQL Managed Instance is the intelligent, scalable cloud database service that combines the broadest SQL Server database engine compatibility with all the benefits of a fully managed and evergreen platform as a service.

The goal of the high availability architecture in SQL Managed Instance is to guarantee that your database is up and running without worrying about the impact of maintenance operations and outages. This solution is designed to:

- Ensure that committed data is never lost because of failures.
- Ensure that maintenance failures don't affect your workload.
- Ensure that the database won't be a single point of failure in your software architecture.

For more information about how Azure SQL Managed Instance supports operational excellence for your application workloads, reference the following articles:

- Overview of Azure SQL Managed Instance management operations
- Monitoring Azure SQL Managed Instance management operations

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure SQL Managed Instance, and operational excellence.

Design considerations

Azure SQL Managed Instance includes the following design considerations:

- Define an application performance SLA and monitor it with alerts. Detecting quickly when your application performance inadvertently degrades below an acceptable level is important to maintain high resiliency. Use a monitoring solution to set alerts on key query performance metrics so you can take action when the performance breaks the SLA.
- Use point-in-time restore to recover from human error. Point-in-time restore returns your database to an earlier point in time to recover data from changes done inadvertently. For more information, read the Point-in-time-restore (PITR) documentation for managed instance.

- Use geo-restore to recover from a service outage. Geo-restore restores a database from a geo-redundant backup into a managed instance in a different region. For more information, reference Recover a database using Geo-restore documentation.
- Consider the time required for certain operations. Make sure you separate time to thoroughly test the amount of time required to scale up and down your existing managed instance, and to create a new managed instance. This timing practice ensures that you understand completely how time consuming operations will affect your RTO and RPO.

Checklist

Have you configured Azure SQL Managed Instance with operational excellence in mind?

- ✓ Use the Business Critical Tier.
- ✓ Configure a secondary instance and an Autofailover group to enable failover to another region.
- ✓ Implement Retry Logic.
- ✓ Monitor your SQL MI instance in near-real time to detect reliability incidents.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure SQL Managed Instance configuration for operational excellence:

Recommendation	Description
Use the Business Critical Tier.	This tier provides higher resiliency to failures and faster failover times because of the underlying HA architecture, among other benefits. For more information, reference SQL Managed Instance High availability.
Configure a secondary instance and an Autofailover group to enable failover to another region.	If an outage impacts one or more of the databases in the managed instance, you can manually or automatically failover all the databases inside the instance to a secondary region. For more information, read the Autofailover groups documentation for managed instance.
Implement Retry Logic.	Although Azure SQL MI is resilient to transitive infrastructure failures, these failures might affect your connectivity. When a transient error occurs while working with SQL MI, make sure your code can retry the call. For more information, reference how to implement retry logic.

Recommendation	Description
Monitor your SQL MI instance in near-real time to detect reliability incidents.	Use one of the available solutions to monitor your SQL MI to detect potential reliability incidents early and make your databases more reliable. Choose a near real-time monitoring solution to quickly react to incidents. For more information, check out the Azure SQL Managed Instance monitoring options .

Next step

Azure Cosmos DB and reliability

Azure Cosmos DB and reliability

Article • 11/30/2022

Azure Cosmos DB is a fully managed NoSQL database for modern app development.

Key features include:

- Guaranteed speed at any scale
- Simplified application development
- Mission-critical ready
- Fully managed and cost effective

To understand how Azure Cosmos DB bolsters resiliency for your application workload, reference the following articles:

- Distribute your data globally with Azure Cosmos DB
- How does Azure Cosmos DB provide high availability
- Consistency levels in Azure Cosmos DB
- Configure Azure Cosmos DB account with periodic backup

The following sections include design considerations, a configuration checklist, recommended configuration options, and source artifacts specific to Azure Cosmos DB and reliability.

Design considerations

Azure Cosmos DB includes the following design considerations:

- SLA for read availability for Database Accounts spanning two or more Azure regions.
- SLAs for throughput, consistency, availability, and latency.
- SLA for both read and write availability with the configuration of multiple Azure regions as writable endpoints.

For more detailed information about SLAs specific to this product, see Azure Cosmos DB Service Level Agreements.

Checklist

Have you configured Azure Cosmos DB with reliability in mind?

- ✓ Deploy Azure Cosmos DB and the application in the region that corresponds to end users.
- ✓ If the multi-master option is enabled on Azure Cosmos DB, it's important to understand Conflict Types and Resolution Policies.
- ✓ Start with, Session, the default consistency level.
- ✓ Change the consistency level, depending on the data operation and usage.
- ✓ Evaluate connectivity modes and connection protocols in Azure Cosmos DB.
- ✓ Configure preferred locations.
- ✓ Specify index precisions.
- ✓ Use Azure Monitor to see the provisioned autoscale max `RU/s` (Autoscale Max Throughput) and the `RU/s` the system is currently scaled to (Provisioned Throughput).
- ✓ Understand your traffic pattern to pick the right option for provisioned throughput types.
- ✓ *New applications*: If you don't know your traffic pattern yet, start at the entry point `RU/s` to avoid over-provisioning in the beginning.
- ✓ *Existing applications*: Use Azure Monitor metrics to determine if your traffic pattern is suitable for autoscale.
- ✓ *Existing applications*: Find the normalized request unit consumption metric of your database or container.
- ✓ *Existing applications*: The closer the number is to `100%`, the more you're fully using your provisioned `RU/s`.
- ✓ Set provisioned `RU/s` to `T` for all hours in a month.
- ✓ Enable automatic failover when you configure Azure Cosmos DB accounts used for production workloads.
- ✓ Implement retry logic in your client.
- ✓ For query-intensive workloads, use Windows `64-bit` instead of Linux or Windows `32-bit` host processing.
- ✓ To reduce latency and CPU jitter, enable accelerated networking on client virtual machines in both Windows and Linux.
- ✓ Increase the number of threads and tasks.
- ✓ To avoid network latency, collocate client in the same region as Azure Cosmos DB.
- ✓ Call OpenAsync to avoid startup latency on first request.
- ✓ Scale out client applications across multiple servers if client consumes more than `50,000` `RU/s`.
- ✓ Select a partition key.
- ✓ Ensure the partition key is a property that has a value that doesn't change.
- ✓ You can't change a partition key after it's created with the collection.
- ✓ Ensure the partition key has a high cardinality.

- ✓ Ensure the partition key spreads RU consumption and data storage evenly across all logical partitions.
- ✓ Ensure you're running read queries with the partitioned column to reduce RU consumption and latency.
- ✓ Evaluate ways to improve data performance.
- ✓ Configure data replication to ensure Azure Cosmos DB meets the SLAs.

Configuration recommendations

Explore the following table of recommendations to optimize your Azure Cosmos DB configuration for reliability:

Recommendation	Description
Deploy Azure Cosmos DB and the application in the region that corresponds to end users.	There are two common scenarios for configuring two or more regions: Delivering low-latency access to data to end users no matter where they're located around the globe. Adding regional resiliency for business continuity and disaster recovery (BCDR). For delivering low-latency to end users, it's recommended to deploy both the application and add Azure Cosmos DB in the regions that correspond to where the application's users are located.
Start with, Session, the default consistency level.	It's the recommended consistency level to start with as it receives data later, but in the same order as the writes.
Change the consistency level, depending on the data operation and usage.	Depending on the type of data stored, different consistency levels may be changed on a per request basis. For example, if log data is written to Azure Cosmos DB, an **Eventual** consistency may be relevant, but if writing e-commerce transactions, then **Strong** may be more appropriate.
Evaluate connectivity modes and connection protocols in Azure Cosmos DB.	Azure Cosmos DB supports two connectivity modes. In *Gateway mode*, requests are always made to the Azure Cosmos DB gateway, which forwards it to the corresponding data partitions. In *Direct connectivity mode*, the client fetches the mapping of tables to partitions, and requests are made directly against data partitions. We recommend Direct, the default mode. Azure Cosmos DB supports two connection protocols: `HTTPS` and `TCP`, which is the default. `TCP` is recommended because it's more lightweight.

Recommendation	Description
Configure preferred locations.	Setting preferred locations can improve query performance. To take advantage of global distribution, client applications can specify the ordered preference list of regions to be used to perform document operations, which can be done by setting the connection policy. Based on the Azure Cosmos DB account configuration, current regional availability and the preference list specified, the most optimal endpoint will be chosen by the SQL SDK to perform write and read operations. This preference list is specified when initializing a connection using the SQL SDKs. The SDKs accept an optional parameter, `PreferredLocations`, that is an ordered list of Azure regions. The SDK will automatically send all writes to the current write region. All reads will be sent to the first available region in the `PreferredLocations` list. If the request fails, the client will fail down the list to the next region, and so on. The SDKs will only attempt to read from the regions specified in `PreferredLocations`.
Specify index precisions.	Setting these values appropriately can improve query performance and reduce throughput requests. You can use index precision to make trade-offs between index storage overhead and query performance. For numbers, we recommend using the default precision configuration of `-1`(maximum). Because numbers are `8` bytes in JSON, this is equivalent to a configuration of `8` bytes. Choosing a lower value for precision, such as `1` through `7`, means that values within some ranges map to the same index entry. You reduce index storage space, but query execution might have to process more documents. It consumes more throughput in request units. Index precision configuration has more practical application with string ranges. Because strings can be any arbitrary length, the choice of the index precision might affect the performance of string range queries. It also may affect the amount of index storage space that's required. String Range indexes can be configured with `1` through `100` or `-1` (maximum).
Existing applications: Find the normalized request unit consumption metric of your database or container.	*Normalized* usage is a measure of how much you're currently using your standard (manual) provisioned throughput.
Set provisioned `RU/s` to `T` for all hours in a month.	If you set provisioned `RU/s` to `T` and use the full amount for `66%` of the hours or more, it's estimated you'll save with standard (manual) provisioned `RU/s`. If you set autoscale max `RU/s` to `Tmax` and use the full amount `Tmax` for `66%` of the hours or less, it's estimated you'll save with autoscale.

Recommendation	Description
Scale out client applications across multiple servers if client consumes more than `50,000` `RU/s`.	There could be a bottleneck because of the machine capping out on CPU or network usage.
Ensure the partition key spreads RU consumption and data storage evenly across all logical partitions.	This spread ensures even RU consumption and storage distribution across your physical partitions.
Evaluate ways to improve data performance.	Best practices for query performance: - Connection policy: Use direct connection mode. - Connection Policy: Use the `TCP` protocol `Call OpenAsync` to avoid startup latency on first request. - Collocate clients in the same Azure region for performance. - Increase number of threads and tasks. - Install the most recent SDK. - Use a singleton Azure Cosmos DB client for the lifetime of your application. - Increase `System.Net` MaxConnections per host when using Gateway mode. - Tune parallel queries for partitioned collections. - Turn on server-side GC. - Implement backoff at `RetryAfter` intervals. - Scale out your client-workload. - Cache document URIs for lower read latency. - Tune the page size for queries and read feeds for better performance. - Use 64-bit host processing. - Exclude unused paths from indexing for faster writes. - Measure and tune for lower request units and second usage. - Handle rate limiting and request rates that are too large. - Design for smaller documents for higher throughput.
Configure data replication to ensure Azure Cosmos DB meets the SLAs	If you've replicated your data in more than one data center, Azure Cosmos DB automatically rolls over your operations should a regional data center go offline. You can create a prioritized list of failover regions using the regions in which your data is replicated. Even within a single data center, Azure Cosmos DB automatically replicates data for high availability giving you the choice of consistency levels.

Source artifacts

To check for `cosmosdb` instances where automatic failover isn't enabled, use the following query:

```SQL
Resources
|where  type =~ 'Microsoft.DocumentDb/databaseAccounts'
|where properties.enableAutomaticFailover!=True
```

Use the following query to see the list of multiregion writes:

```SQL
resources
| where type == "microsoft.documentdb/databaseaccounts"
 and properties.enableMultipleWriteLocations == "true"
```

To view consistency levels for your Azure Cosmos DB accounts, use the following query:

```SQL
Resources
| project name, type, location, consistencyLevel =
properties.consistencyPolicy.defaultConsistencyLevel
| where type == "microsoft.documentdb/databaseaccounts"
| order by name asc
```

To check if multilocation isn't selected, use the following query:

```SQL
Resources
|where  type =~ 'Microsoft.DocumentDb/databaseAccounts'
|where array_length( properties.locations) <=1
```

Learn more

- High availability in Azure Cosmos DB
- Autoscale FAQ
- Performance tips for Azure Cosmos DB

Next step

Azure Cosmos DB and operational excellence

Azure Cosmos DB and operational excellence

Article • 11/30/2022

Azure Cosmos DB is a fully managed NoSQL database for modern app development.

Key features include:

- Guaranteed speed at any scale
- Simplified application development
- Mission-critical ready
- Fully managed and cost effective

To understand how Azure Cosmos DB promotes operational excellence for your application workload, reference the following articles:

- Monitor Azure Cosmos DB
- Monitor and debug with insights in Azure Cosmos DB
- Visualize Azure Cosmos DB data by using the Power BI connector

The following sections include design considerations, a configuration checklist, recommended configuration options, and source artifacts specific to Azure Cosmos DB, and operational excellence.

Design considerations

Azure Cosmos DB includes the following design considerations:

- SLA for read availability for Database Accounts spanning two or more Azure regions.
- SLAs for throughput, consistency, availability, and latency.
- SLA for both read and write availability with the configuration of multiple Azure regions as writable endpoints.

For more granular information specific to this product, reference Azure Cosmos DB Service Level Agreements.

Checklist

Have you configured Azure Cosmos DB with operational excellence in mind?

- ✓ Monitor for normal and abnormal activity.
- ✓ If the multi-master option is enabled on Azure Cosmos DB, it's important to understand Conflict Types and Resolution Policies.
- ✓ Start with, Session, the default consistency level.
- ✓ Use Azure Monitor to see the provisioned autoscale max `RU/s` (Autoscale Max Throughput) and the `RU/s` the system is currently scaled to (Provisioned Throughput).
- ✓ Understand your traffic pattern to pick the right option for provisioned throughput types.
- ✓ *New applications*: If you don't know your traffic pattern yet, start at the entry point `RU/s` to avoid over-provisioning in the beginning.
- ✓ *Existing applications*: Use Azure Monitor metrics to determine if your traffic pattern is suitable for autoscale.
- ✓ *Existing applications*: Find the normalized request unit consumption metric of your database or container.
- ✓ *Existing applications*: The closer the number is to `100%`, the more you're fully using your provisioned `RU/s`.
- ✓ Set provisioned `RU/s` to `T` for all hours in a month.
- ✓ Enable automatic failover when you configure Azure Cosmos DB accounts used for production workloads.
- ✓ Implement retry logic in your client.
- ✓ For query-intensive workloads, use Windows `64-bit` instead of Linux or Windows `32-bit` host processing.
- ✓ To reduce latency and CPU jitter, enable accelerated networking on client virtual machines in both Windows and Linux.
- ✓ Increase the number of threads and tasks.
- ✓ To avoid network latency, colocate the client in the same region as the Azure Cosmos DB instance.
- ✓ Call OpenAsync to avoid startup latency on first request.
- ✓ Scale out client applications across multiple servers if client consumes more than `50,000` `RU/s`.
- ✓ Select a partition key.
- ✓ Ensure the partition key is a property that has a value that doesn't change.
- ✓ You can't change a partition key after it's created with the collection.
- ✓ Ensure the partition key has a high cardinality.
- ✓ Ensure the partition key spreads RU consumption and data storage evenly across all logical partitions.
- ✓ Ensure you're running read queries with the partitioned column to reduce RU consumption and latency.

Configuration recommendations

Explore the following table of recommendations to optimize operational excellence for your Azure Cosmos DB configuration:

Recommendation	Description
Monitor for normal and abnormal activity.	The Azure Activity Log is a subscription log that provides insight into subscription-level events that have occurred in Azure. The Activity Log reports control plane events for your subscriptions under the Administrative category. Using the Activity Log, you can determine the *what*, *who*, and *when* for any write operations (`PUT`, `POST`, `DELETE`) taken on the resources in your subscription. You can also understand the status of the operation and other relevant properties. The Activity Log differs from Diagnostic Logs. Activity Logs provide data about the operations on a resource from the outside (the *control plane*). In the Azure Cosmos DB context, some of the control plane operations include create collection, list keys, delete keys, list database, and more. Diagnostic Logs are emitted by a resource and provide information about the operation of that resource (the *data plane*). Some of the data plane diagnostic log examples include delete, insert, ReadFeed operation, and more.
Start with, Session, the default consistency level.	It's the recommended consistency level to start with as it receives data later, but in the same order as the writes.
Existing applications: Find the normalized request unit consumption metric of your database or container.	*Normalized* usage is a measure of how much you're currently using your standard (manual) provisioned throughput.
Set provisioned `RU/s` to `T` for all hours in a month.	If you set provisioned `RU/s` to `T` and use the full amount for `66%` of the hours or more, it's estimated you'll save with standard (manual) provisioned `RU/s`. If you set autoscale max `RU/s` to `Tmax` and use the full amount `Tmax` for `66%` of the hours or less, it's estimated you'll save with autoscale.
Scale out client applications across multiple servers if client consumes more than `50,000` `RU/s`.	There could be a bottleneck because of the machine capping out on CPU or network usage.

Recommendation	Description
Ensure the partition key spreads RU consumption and data storage evenly across all logical partitions.	This spread ensures even RU consumption and storage distribution across your physical partitions.

Source artifacts

To check for `cosmosdb` instances where automatic failover isn't enabled, use the following query:

```sql
Resources
|where  type =~ 'Microsoft.DocumentDb/databaseAccounts'
|where properties.enableAutomaticFailover!=True
```

Use the following query to see the list of multiregion writes:

```sql
resources
| where type == "microsoft.documentdb/databaseaccounts"
 and properties.enableMultipleWriteLocations == "true"
```

To view consistency levels for your Azure Cosmos DB accounts, use the following query:

```sql
Resources
| project name, type, location, consistencyLevel =
properties.consistencyPolicy.defaultConsistencyLevel
| where type == "microsoft.documentdb/databaseaccounts"
| order by name asc
```

To check if multilocation isn't selected, use the following query:

```sql
Resources
|where  type =~ 'Microsoft.DocumentDb/databaseAccounts'
|where array_length( properties.locations) <=1
```

Learn more

- Autoscale FAQ
- Performance tips for Azure Cosmos DB

Next step

Azure Stack Hub and reliability

Azure Stack Hub and reliability

Article • 11/30/2022

Azure Stack Hub is a hybrid cloud platform that lets you provide Azure services from your datacenter. It provides a way to run apps in an on-premises environment.

This service unlocks the following hybrid cloud use cases for customer-facing and internal line-of-business apps:

- *Edge and disconnected solutions*: Addresses latency and connectivity requirements by processing data locally.
- *Cloud apps that meet varied regulations*: Allows you to develop and deploy apps with full flexibility to meet regulatory or policy requirements.
- *Cloud app model on-premises*: Provides Azure services, containers, serverless, and microservice architectures to update and extend existing apps or build new ones.

For more information, reference Azure Stack Hub overview.

To understand how Azure Stack Hub supports resiliency for your application workload, reference the following articles:

- Capacity planning for Azure Stack Hub overview
- Storage Spaces Direct cache and capacity tiers
- Datacenter integration planning considerations for Azure Stack Hub integrated systems

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Stack Hub and reliability.

Design considerations

Azure Stack Hub includes the following design considerations:

- Microsoft doesn't provide an SLA for Azure Stack Hub because Microsoft doesn't have control over customer datacenter reliability, people, and processes.
- Azure Stack Hub only supports a single Scale Unit (SU) within a single region, which consists of between four and 16 servers that use Hyper-V failover clustering. Each region serves as an independent Azure Stack Hub *stamp* with separate portal and API endpoints.
- Azure Stack Hub doesn't support Availability Zones because it consists of a single *region* or a single physical location. High availability to cope with outages of a

single location should be implemented by using two Azure Stack Hub instances deployed in different physical locations.
- Azure Stack Hub supports premium storage to ensure compatibility. However, provisioning premium storage accounts or disks doesn't guarantee that storage objects will be allocated onto SSD or NVMe drives.
- Azure Stack Hub supports only a subset of VPN Gateway SKUs available in Azure with a limited bandwidth of `100` or `200 Mbps`.
- Only one site-to-site (S2S) VPN connection can be created between two Azure Stack Hub deployments. This connection limit is because of a platform limitation that allows only a single VPN connection to the same IP address. Multiple S2S VPN connections with higher throughput can be established using third-party NVAs.
- Apply general Azure configuration recommendations for all Azure Stack Hub services.

Checklist

Have you configured Azure Stack Hub with reliability in mind?

✓ Treat Azure Stack Hub as a scale unit and deploy multiple instances to remove Azure Stack Hub as a single point of failure for encompassed workloads.

Configuration recommendations

Consider the following recommendation table to optimize your Azure Stack Hub configuration for reliability:

Recommendation	Description
Treat Azure Stack Hub as a scale unit and deploy multiple instances to remove Azure Stack Hub as a single point of failure for encompassed workloads.	Deploy workloads in either an active-active or active-passive configuration across Azure Stack Hub stamps or Azure.

Next step

Azure Stack Hub and operational excellence

Azure Stack Hub and operational excellence

Article • 11/30/2022

Azure Stack Hub is a hybrid cloud platform that lets you provide Azure services from your datacenter. It provides a way to run apps in an on-premises environment.

This service unlocks the following hybrid cloud use cases for customer-facing and internal line-of-business apps:

- *Edge and disconnected solutions*: Addresses latency and connectivity requirements by processing data locally.
- *Cloud apps that meet varied regulations*: Allows you to develop and deploy apps with full flexibility to meet regulatory or policy requirements.
- *Cloud app model on-premises*: Provides Azure services, containers, serverless, and microservice architectures to update and extend existing apps or build new ones.

For more information, reference Azure Stack Hub overview.

To understand how Azure Stack Hub supports operational excellence for your application workload, reference the following articles:

- Monitor health and alerts in Azure Stack Hub
- Monitor Azure Stack Hub hardware components
- Manage network resources in Azure Stack Hub

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure Stack Hub and operational excellence.

Design considerations

Azure Stack Hub includes the following design considerations:

- Microsoft doesn't provide an SLA for Azure Stack Hub because Microsoft doesn't have control over customer datacenter reliability, people, and processes.
- Azure Stack Hub only supports a single Scale Unit (SU) within a single region, which consists of between four and 16 servers that use Hyper-V failover clustering. Each region serves as an independent Azure Stack Hub *stamp* with separate portal and API endpoints.

- Azure Stack Hub doesn't support Availability Zones because it consists of a single *region* or a single physical location. High availability to cope with outages of a single location should be implemented by using two Azure Stack Hub instances deployed in different physical locations.
- Apply general Azure configuration recommendations for all Azure Stack Hub services.

Checklist

Have you configured Azure Stack Hub with operational excellence in mind?

✓ Treat Azure Stack Hub as a scale unit and deploy multiple instances to remove Azure Stack Hub as a single point of failure for encompassed workloads.

Configuration recommendations

Consider the following recommendation table to optimize your Azure Stack Hub configuration for operational excellence:

Recommendation	Description
Treat Azure Stack Hub as a scale unit and deploy multiple instances to remove Azure Stack Hub as a single point of failure for encompassed workloads.	Deploy workloads in either an active-active or active-passive configuration across Azure Stack Hub stamps or Azure.

Next step

Storage Accounts and reliability

Storage Accounts and reliability

Article • 11/30/2022

Azure Storage Accounts are ideal for workloads that require fast and consistent response times, or that have a high number of input output (IOP) operations per second. Storage accounts contain all your Azure Storage data objects, which include:

- Blobs
- File shares
- Queues
- Tables
- Disks

Storage accounts provide a unique namespace for your data that's accessible anywhere over `HTTP` or `HTTPS`.

For more information about the different types of storage accounts that support different features, reference Types of storage accounts.

To understand how an Azure storage account supports resiliency for your application workload, reference the following articles:

- Azure storage redundancy
- Disaster recovery and storage account failover

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure storage accounts and reliability.

Design considerations

Azure storage accounts include the following design considerations:

- General purpose v1 storage accounts provide access to all Azure Storage services, but may not have the latest features or the lower per-gigabyte pricing. It's recommended to use general purpose v2 storage accounts, in most cases. Reasons to use v1 include:
    - Applications require the classic deployment model.
    - Applications are transaction intensive or use significant geo-replication bandwidth, but don't require large capacity.
    - The use of a Storage Service REST API that is earlier than February 14, 2014, or a client library with a version earlier than `4.x` is required. An application upgrade isn't possible.

For more information, reference the Storage account overview.

- Storage account names must be between three and 24 characters and may contain numbers, and lowercase letters only.
- For current SLA specifications, reference SLA for Storage Accounts .
- Go to Azure Storage redundancy to determine which redundancy option is best for a specific scenario.
- Storage account names must be unique within Azure. No two storage accounts can have the same name.

Checklist

Have you configured your Azure Storage Account with reliability in mind?

- ✓ Turn on soft delete for blob data.
- ✓ Use Azure AD to authorize access to blob data.
- ✓ Consider the principle of least privilege when you assign permissions to an Azure AD security principal through Azure RBAC.
- ✓ Use managed identities to access blob and queue data.
- ✓ Use blob versioning or immutable blobs to store business-critical data.
- ✓ Restrict default internet access for storage accounts.
- ✓ Enable firewall rules.
- ✓ Limit network access to specific networks.
- ✓ Allow trusted Microsoft services to access the storage account.
- ✓ Enable the **Secure transfer required** option on all your storage accounts.
- ✓ Limit shared access signature (SAS) tokens to `HTTPS` connections only.
- ✓ Avoid and prevent using Shared Key authorization to access storage accounts.
- ✓ Regenerate your account keys periodically.
- ✓ Create a revocation plan and have it in place for any SAS that you issue to clients.
- ✓ Use near-term expiration times on an impromptu SAS, service SAS, or account SAS.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your Azure Storage Account:

Recommendation	Description
Turn on soft delete for blob data.	Soft delete for Azure Storage blobs enables you to recover blob data after it has been deleted.

Recommendation	Description
Use Azure AD to authorize access to blob data.	Azure AD provides superior security and ease of use over Shared Key for authorizing requests to blob storage. It's recommended to use Azure AD authorization with your blob and queue applications when possible to minimize potential security vulnerabilities inherent in Shared Key. For more information, reference Authorize access to Azure blobs and queues using Azure Active Directory.
Consider the principle of least privilege when you assign permissions to an Azure AD security principal through Azure RBAC.	When assigning a role to a user, group, or application, grant that security principal only those permissions necessary for them to perform their tasks. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.
Use managed identities to access blob and queue data.	Azure Blob and Queue storage support Azure AD authentication with managed identities for Azure resources. Managed identities for Azure resources can authorize access to blob and queue data using Azure AD credentials from applications running in Azure virtual machines (VMs), function apps, virtual machine scale sets, and other services. By using managed identities for Azure resources together with Azure AD authentication, you can avoid storing credentials with your applications that run in the cloud and issues with expiring service principals. Reference Authorize access to blob and queue data with managed identities for Azure resources for more information.
Use blob versioning or immutable blobs to store business-critical data.	Consider using Blob versioning to maintain previous versions of an object or the use of legal holds and time-based retention policies to store blob data in a WORM (Write Once, Read Many) state. Immutable blobs can be read, but can't be modified or deleted during the retention interval. For more information, reference Store business-critical blob data with immutable storage.
Restrict default internet access for storage accounts.	By default, network access to Storage Accounts isn't restricted and is open to all traffic coming from the internet. Access to storage accounts should be granted to specific Azure Virtual Networks only whenever possible or use private endpoints to allow clients on a virtual network (VNet) to access data securely over a Private Link. Reference Use private endpoints for Azure Storage for more information. Exceptions can be made for Storage Accounts that need to be accessible over the internet.
Enable firewall rules.	Configure firewall rules to limit access to your storage account to requests that originate from specified IP addresses or ranges, or from a list of subnets in an Azure Virtual Network (VNet). For more information about configuring firewall rules, reference Configure Azure Storage firewalls and virtual networks.

Recommendation	Description
Limit network access to specific networks.	Limiting network access to networks hosting clients requiring access reduces the exposure of your resources to network attacks either by using the built-in Firewall and virtual networks functionality or by using private endpoints.
Allow trusted Microsoft services to access the storage account.	Turning on firewall rules for storage accounts blocks incoming requests for data by default, unless the requests originate from a service operating within an Azure Virtual Network (VNet) or from allowed public IP addresses. Blocked requests include those requests from other Azure services, from the Azure portal, from logging and metrics services, and so on. You can permit requests from other Azure services by adding an exception to allow trusted Microsoft services to access the storage account. For more information about adding an exception for trusted Microsoft services, reference Configure Azure Storage firewalls and virtual networks.
Enable the **Secure transfer required** option on all your storage accounts.	When you enable the **Secure transfer required** option, all requests made against the storage account must take place over secure connections. Any requests made over HTTP will fail. For more information, reference Require secure transfer in Azure Storage.
Limit shared access signature (SAS) tokens to `HTTPS` connections only.	Requiring `HTTPS` when a client uses a SAS token to access blob data helps to minimize the risk of eavesdropping. For more information, reference Grant limited access to Azure Storage resources using shared access signatures (SAS).
Avoid and prevent using Shared Key authorization to access storage accounts.	It's recommended to use Azure AD to authorize requests to Azure Storage and to prevent Shared Key Authorization. For scenarios that require Shared Key authorization, always prefer SAS tokens over distributing the Shared Key.
Regenerate your account keys periodically.	Rotating the account keys periodically reduces the risk of exposing your data to malicious actors.
Create a revocation plan and have it in place for any SAS that you issue to clients.	If a SAS is compromised, you'll want to revoke that SAS immediately. To revoke a user delegation SAS, revoke the user delegation key to quickly invalidate all signatures associated with that key. To revoke a service SAS that's associated with a stored access policy, you can delete the stored access policy, rename the policy, or change its expiry time to a time that is in the past.
Use near-term expiration times on an impromptu SAS, service SAS, or account SAS.	If a SAS is compromised, it's valid only for a short time. This practice is especially important if you can't reference a stored access policy. Near-term expiration times also limit the amount of data that can be written to a blob by limiting the time available to upload to it. Clients should renew the SAS well before the expiration to allow time for retries if the service providing the SAS is unavailable.

Next step

Storage Accounts and security

Storage Accounts and security

Article • 11/30/2022

Azure Storage Accounts are ideal for workloads that require fast and consistent response times, or that have a high number of input output (IOP) operations per second. Storage accounts contain all your Azure Storage data objects, which include:

- Blobs
- File shares
- Queues
- Tables
- Disks

Storage accounts provide a unique namespace for your data that's accessible anywhere over `HTTP` or `HTTPS`.

For more information about the different types of storage accounts that support different features, reference Types of storage accounts.

To understand how an Azure storage account boosts security for your application workload, reference the following articles:

- Azure security baseline for Azure Storage
- Azure Storage encryption for data at rest
- Use private endpoints for Azure Storage

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure storage accounts and security.

Design considerations

Azure storage accounts include the following design considerations:

- Storage account names must be between three and 24 characters and may contain numbers, and lowercase letters only.
- For current SLA specifications, reference SLA for Storage Accounts.
- Go to Azure Storage redundancy to determine which redundancy option is best for a specific scenario.
- Storage account names must be unique within Azure. No two storage accounts can have the same name.

Checklist

Have you configured your Azure Storage Account with security in mind?

- ✓ Enable Azure Defender for all your storage accounts.
- ✓ Turn on soft delete for blob data.
- ✓ Use Azure AD to authorize access to blob data.
- ✓ Consider the principle of least privilege when you assign permissions to an Azure AD security principal through Azure RBAC.
- ✓ Use managed identities to access blob and queue data.
- ✓ Use blob versioning or immutable blobs to store business-critical data.
- ✓ Restrict default internet access for storage accounts.
- ✓ Enable firewall rules.
- ✓ Limit network access to specific networks.
- ✓ Allow trusted Microsoft services to access the storage account.
- ✓ Enable the **Secure transfer required** option on all your storage accounts.
- ✓ Limit shared access signature (SAS) tokens to `HTTPS` connections only.
- ✓ Avoid and prevent using Shared Key authorization to access storage accounts.
- ✓ Regenerate your account keys periodically.
- ✓ Create a revocation plan and have it in place for any SAS that you issue to clients.
- ✓ Use near-term expiration times on an impromptu SAS, service SAS, or account SAS.

Configuration recommendations

Consider the following recommendations to optimize security when configuring your Azure Storage Account:

Recommendation	Description
Enable Azure Defender for all your storage accounts.	Azure Defender for Azure Storage provides an extra layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. Security alerts are triggered in Azure Security Center when anomalies in activity occur. Alerts are also sent through email to subscription administrators, with details of suspicious activity and recommendations on how to investigate, and remediate threats. For more information, reference Configure Azure Defender for Azure Storage.
Turn on soft delete for blob data.	Soft delete for Azure Storage blobs enables you to recover blob data after it has been deleted.

Recommendation	Description
Use Azure AD to authorize access to blob data.	Azure AD provides superior security and ease of use over Shared Key for authorizing requests to blob storage. It's recommended to use Azure AD authorization with your blob and queue applications when possible to minimize potential security vulnerabilities inherent in Shared Key. For more information, reference Authorize access to Azure blobs and queues using Azure Active Directory.
Consider the principle of least privilege when you assign permissions to an Azure AD security principal through Azure RBAC.	When assigning a role to a user, group, or application, grant that security principal only those permissions necessary for them to complete their tasks. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.
Use managed identities to access blob and queue data.	Azure Blob and Queue storage support Azure AD authentication with managed identities for Azure resources. Managed identities for Azure resources can authorize access to blob and queue data using Azure AD credentials from applications running in Azure virtual machines (VMs), function apps, virtual machine scale sets, and other services. By using managed identities for Azure resources together with Azure AD authentication, you can avoid storing credentials with your applications that run in the cloud and issues with expiring service principals. Reference Authorize access to blob and queue data with managed identities for Azure resources for more information.
Use blob versioning or immutable blobs to store business-critical data.	Consider using Blob versioning to maintain previous versions of an object or the use of legal holds and time-based retention policies to store blob data in a WORM (Write Once, Read Many) state. Immutable blobs can be read, but can't be modified or deleted during the retention interval. For more information, reference Store business-critical blob data with immutable storage.
Restrict default internet access for storage accounts.	By default, network access to Storage Accounts isn't restricted and is open to all traffic coming from the internet. Access to storage accounts should be granted to specific Azure Virtual Networks only whenever possible or use private endpoints to allow clients on a virtual network (VNet) to access data securely over a Private Link. Reference Use private endpoints for Azure Storage for more information. Exceptions can be made for Storage Accounts that need to be accessible over the internet.
Enable firewall rules.	Configure firewall rules to limit access to your storage account to requests that originate from specified IP addresses or ranges, or from a list of subnets in an Azure Virtual Network (VNet). For more information about configuring firewall rules, reference Configure Azure Storage firewalls and virtual networks.

Recommendation	Description
Limit network access to specific networks.	Limiting network access to networks hosting clients requiring access reduces the exposure of your resources to network attacks either by using the built-in Firewall and virtual networks functionality or by using private endpoints.
Allow trusted Microsoft services to access the storage account.	Turning on firewall rules for storage accounts blocks incoming requests for data by default, unless the requests originate from a service operating within an Azure Virtual Network (VNet) or from allowed public IP addresses. Blocked requests include those requests from other Azure services, from the Azure portal, from logging and metrics services, and so on. You can permit requests from other Azure services by adding an exception to allow trusted Microsoft services to access the storage account. For more information about adding an exception for trusted Microsoft services, reference Configure Azure Storage firewalls and virtual networks.
Enable the **Secure transfer required** option on all your storage accounts.	When you enable the **Secure transfer required** option, all requests made against the storage account must take place over secure connections. Any requests made over HTTP will fail. For more information, reference Require secure transfer in Azure Storage.
Limit shared access signature (SAS) tokens to `HTTPS` connections only.	Requiring `HTTPS` when a client uses a SAS token to access blob data helps to minimize the risk of eavesdropping. For more information, reference Grant limited access to Azure Storage resources using shared access signatures (SAS).
Avoid and prevent using Shared Key authorization to access storage accounts.	It's recommended to use Azure AD to authorize requests to Azure Storage and to prevent Shared Key Authorization. For scenarios that require Shared Key authorization, always prefer SAS tokens over distributing the Shared Key.
Regenerate your account keys periodically.	Rotating the account keys periodically reduces the risk of exposing your data to malicious actors.
Create a revocation plan and have it in place for any SAS that you issue to clients.	If a SAS is compromised, you'll want to revoke that SAS immediately. To revoke a user delegation SAS, revoke the user delegation key to quickly invalidate all signatures associated with that key. To revoke a service SAS that's associated with a stored access policy, you can delete the stored access policy, rename the policy, or change its expiry time to a time that is in the past.
Use near-term expiration times on an impromptu SAS, service SAS, or account SAS.	If a SAS is compromised, it's valid only for a short time. This practice is especially important if you can't reference a stored access policy. Near-term expiration times also limit the amount of data that can be written to a blob by limiting the time available to upload to it. Clients should renew the SAS well before the expiration to allow time for retries if the service providing the SAS is unavailable.

Next step

Storage Accounts and cost optimization

Storage Accounts and cost optimization

Article • 11/30/2022

Azure Storage Accounts are ideal for workloads that require fast and consistent response times, or that have a high number of input output (IOP) operations per second. Storage accounts contain all your Azure Storage data objects, which include:

- Blobs
- File shares
- Queues
- Tables
- Disks

Storage accounts provide a unique namespace for your data that's accessible anywhere over `HTTP` or `HTTPS`.

For more information about the different types of storage accounts that support different features, reference Types of storage accounts.

To understand how an Azure storage account can optimize costs for your workload, reference the following articles:

- Plan and manage costs for Azure Blob Storage
- Optimize costs for Blob storage with reserved capacity
- Understand how reservation discounts are applied to Azure storage services

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure storage accounts and cost optimization.

Design considerations

Azure storage accounts include the following design considerations:

- Periodically dispose and clean up unused storage resources, such as unattached disks and old snapshots.
- Consider Azure Blob access time tracking and access time-based lifecycle management.
- Transition your data from a hotter access tier to a cooler access tier if there's no access for a period.
- Delete your data if there's no access for an extended period.

Considerations	Description
Periodically dispose and clean up unused storage resources, such as unattached disks and old snapshots.	Unused storage resources can incur cost and it's a good idea to regularly perform cleanup to reduce cost.
Consider Azure Blob access time tracking and access time-based lifecycle management.	Minimize your storage cost automatically by setting up a policy based on last access time to: cost-effective backup storage options.
Transition your data from a hotter access tier to a cooler access tier if there's no access for a period	For example: - Hot to cool - Cool to archive - Hot to archive

Checklist

Have you configured your Azure Storage Account with cost optimization in mind?

- ✓ Consider cost savings by reserving data capacity for block blob storage.
- ✓ Organize data into access tiers.
- ✓ Use lifecycle policy to move data between access tiers.

Configuration recommendations

Consider the following recommendations to optimize costs when configuring your Azure Storage Account:

Recommendation	Description
Consider cost savings by reserving data capacity for block blob storage.	Save money by reserving capacity for block blob and for Azure Data Lake Storage gen 2 data in standard storage account when customer commits to one or three years reservation.
Organize data into access tiers.	You can reduce cost by placing blob data into the most cost-effective access tier. Place frequently accessed data in a hot tier, less frequent in a cold or archive tier. Use Premium storage for workloads with high transaction volumes or workloads where latency is critical.

Recommendation	Description
Use lifecycle policy to move data between access tiers.	Lifecycle management policy periodically moves data between tiers. Policies can move data based on rules specified by the user. For example, you can create rules that move blobs to the archive tier if that blob has been modified in 90 days. Unused data can be removed completely using a policy. By creating policies that adjust the access tier of your data, you can design the least expensive storage options for your requirements.

Next step

Storage Accounts and operational excellence

Storage Accounts and operational excellence

Article • 11/30/2022

Azure Storage Accounts are ideal for workloads that require fast and consistent response times, or that have a high number of input output (IOP) operations per second. Storage accounts contain all your Azure Storage data objects, which include:

- Blobs
- File shares
- Queues
- Tables
- Disks

Storage accounts provide a unique namespace for your data that's accessible anywhere over `HTTP` or `HTTPS`.

For more information about the different types of storage accounts that support different features, reference Types of storage accounts.

To understand how an Azure storage account can promote operational excellence for your workload, reference the following articles:

- Best practices for monitoring Azure Blob Storage
- Use Azure Storage analytics to collect logs and metrics data
- Azure Storage analytics logging

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure storage accounts and operational excellence.

Design considerations

Azure storage accounts include the following design considerations:

- General purpose v1 storage accounts provide access to all Azure Storage services, but may not have the latest features or the lower per-gigabyte pricing. It's recommended to use general purpose v2 storage accounts, in most cases. Reasons to use v1 include:
  - Applications require the classic deployment model.

  - Applications are transaction intensive or use significant geo-replication bandwidth, but don't require large capacity.
  - The use of a Storage Service REST API that is earlier than February 14, 2014, or a client library with a version earlier than `4.x` is required. An application upgrade isn't possible.

For more information, reference the Storage account overview.

- Storage account names must be between three and 24 characters and may contain numbers, and lowercase letters only.
- For current SLA specifications, reference SLA for Storage Accounts.
- Go to Azure Storage redundancy to determine which redundancy option is best for a specific scenario.
- Storage account names must be unique within Azure. No two storage accounts can have the same name.

Checklist

Have you configured your Azure Storage Account with operational excellence in mind?

- ✓ Enable Azure Defender for all your storage accounts.
- ✓ Turn on soft delete for blob data.
- ✓ Use Azure AD to authorize access to blob data.
- ✓ Consider the principle of least privilege when you assign permissions to an Azure AD security principal through Azure RBAC.
- ✓ Use managed identities to access blob and queue data.
- ✓ Use blob versioning or immutable blobs to store business-critical data.
- ✓ Restrict default internet access for storage accounts.
- ✓ Enable firewall rules.
- ✓ Limit network access to specific networks.
- ✓ Allow trusted Microsoft services to access the storage account.
- ✓ Enable the **Secure transfer required** option on all your storage accounts.
- ✓ Limit shared access signature (SAS) tokens to `HTTPS` connections only.
- ✓ Avoid and prevent using Shared Key authorization to access storage accounts.
- ✓ Regenerate your account keys periodically.
- ✓ Create a revocation plan and have it in place for any SAS that you issue to clients.
- ✓ Use near-term expiration times on an impromptu SAS, service SAS, or account SAS.

Configuration recommendations

Consider the following recommendations to optimize operational excellence when configuring your Azure Storage Account:

Recommendation	Description
Enable Azure Defender for all your storage accounts.	Azure Defender for Azure Storage provides an extra layer of security intelligence that detects unusual and potentially harmful attempts to access or exploit storage accounts. Security alerts are triggered in Azure Security Center when anomalies in activity occur. Alerts are also sent through email to subscription administrators, with details of suspicious activity and recommendations on how to investigate, and remediate threats. For more information, reference Configure Azure Defender for Azure Storage.
Turn on soft delete for blob data.	Soft delete for Azure Storage blobs enables you to recover blob data after it has been deleted.
Use Azure AD to authorize access to blob data.	Azure AD provides superior security and ease of use over Shared Key for authorizing requests to blob storage. It's recommended to use Azure AD authorization with your blob and queue applications when possible to minimize potential security vulnerabilities inherent in Shared Key. For more information, reference Authorize access to Azure blobs and queues using Azure Active Directory.
Consider the principle of least privilege when you assign permissions to an Azure AD security principal through Azure RBAC.	When assigning a role to a user, group, or application, grant that security principal only those permissions necessary for them to complete their tasks. Limiting access to resources helps prevent both unintentional and malicious misuse of your data.
Use managed identities to access blob and queue data.	Azure Blob and Queue storage support Azure AD authentication with managed identities for Azure resources. Managed identities for Azure resources can authorize access to blob and queue data using Azure AD credentials from applications running in Azure virtual machines (VMs), function apps, virtual machine scale sets, and other services. By using managed identities for Azure resources together with Azure AD authentication, you can avoid storing credentials with your applications that run in the cloud and issues with expiring service principals. Reference Authorize access to blob and queue data with managed identities for Azure resources for more information.
Use blob versioning or immutable blobs to store business-critical data.	Consider using Blob versioning to maintain previous versions of an object or the use of legal holds and time-based retention policies to store blob data in a WORM (Write Once, Read Many) state. Immutable blobs can be read, but can't be modified or deleted during the retention interval. For more information, reference Store business-critical blob data with immutable storage.

Recommendation	Description
Restrict default internet access for storage accounts.	By default, network access to Storage Accounts isn't restricted and is open to all traffic coming from the internet. Access to storage accounts should be granted to specific Azure Virtual Networks only whenever possible or use private endpoints to allow clients on a virtual network (VNet) to access data securely over a Private Link. Reference Use private endpoints for Azure Storage for more information. Exceptions can be made for Storage Accounts that need to be accessible over the internet.
Enable firewall rules.	Configure firewall rules to limit access to your storage account to requests that originate from specified IP addresses or ranges, or from a list of subnets in an Azure Virtual Network (VNet). For more information about configuring firewall rules, reference Configure Azure Storage firewalls and virtual networks.
Limit network access to specific networks.	Limiting network access to networks hosting clients requiring access reduces the exposure of your resources to network attacks either by using the built-in Firewall and virtual networks functionality or by using private endpoints.
Allow trusted Microsoft services to access the storage account.	Turning on firewall rules for storage accounts blocks incoming requests for data by default, unless the requests originate from a service operating within an Azure Virtual Network (VNet) or from allowed public IP addresses. Blocked requests include those requests from other Azure services, from the Azure portal, from logging and metrics services, and so on. You can permit requests from other Azure services by adding an exception to allow trusted Microsoft services to access the storage account. For more information about adding an exception for trusted Microsoft services, reference Configure Azure Storage firewalls and virtual networks.
Enable the **Secure transfer required** option on all your storage accounts.	When you enable the **Secure transfer required** option, all requests made against the storage account must take place over secure connections. Any requests made over HTTP will fail. For more information, reference Require secure transfer in Azure Storage.
Limit shared access signature (SAS) tokens to `HTTPS` connections only.	Requiring `HTTPS` when a client uses a SAS token to access blob data helps to minimize the risk of eavesdropping. For more information, reference Grant limited access to Azure Storage resources using shared access signatures (SAS).
Avoid and prevent using Shared Key authorization to access storage accounts.	It's recommended to use Azure AD to authorize requests to Azure Storage and to prevent Shared Key Authorization. For scenarios that require Shared Key authorization, always prefer SAS tokens over distributing the Shared Key.
Regenerate your account keys periodically.	Rotating the account keys periodically reduces the risk of exposing your data to malicious actors.

Recommendation	Description
Create a revocation plan and have it in place for any SAS that you issue to clients.	If a SAS is compromised, you'll want to revoke that SAS immediately. To revoke a user delegation SAS, revoke the user delegation key to quickly invalidate all signatures associated with that key. To revoke a service SAS that's associated with a stored access policy, you can delete the stored access policy, rename the policy, or change its expiry time to a time that is in the past.
Use near-term expiration times on an impromptu SAS, service SAS, or account SAS.	If a SAS is compromised, it's valid only for a short time. This practice is especially important if you can't reference a stored access policy. Near-term expiration times also limit the amount of data that can be written to a blob by limiting the time available to upload to it. Clients should renew the SAS well before the expiration to allow time for retries if the service providing the SAS is unavailable.

Next step

Disks and cost optimization

Disks and cost optimization

Article • 11/30/2022

Azure managed disks are block-level storage volumes that are managed by Azure and used with Azure Virtual Machines. Managed disks are like a physical disk in an on-premises server, but these disks are virtualized.

Available disk types include:

- Ultra disks
- Premium solid-state drives (SSD)
- Standard SSDs
- Standard hard disk drives (HDD)

For more information about the different types of disks, reference Azure managed disk types.

To understand how Azure managed disks are cost-effective solutions for your workload, reference the following articles:

- Overview of Azure Disk Backup
- Understand how your reservation discount is applied to Azure disk storage
- Reduce costs with Azure Disks Reservation

The following sections include design considerations, a configuration checklist, and recommended configuration options specific to Azure managed disks and cost optimization.

Design considerations

Azure Disks include the following design considerations:

- Use a shared disk for workload, such as SQL server failover cluster instance (FCI), file server for general use (IW workload), and SAP ASCS/SCS.
- Consider selective disk backup and restore for Azure VMs.

Considerations	Description

Considerations	Description
Use a shared disk for workload, such as SQL server failover cluster instance (FCI), file server for general use (IW workload), and SAP ASCS/SCS.	You can use shared disks to enable cost-effective clustering instead of setting up your own shared disks through S2D (Storage Spaces Direct). Sample workloads that would benefit from shared disks include: - SQL Server Failover Cluster Instances (FCI) - Scale-out File Server (SoFS) - File Server for General Use (IW workload) - SAP ASCS/SCS

Checklist

Have you configured your Azure managed disk with cost optimization in mind?

- ✓ Configure data and log files on different disks for database workloads.
- ✓ Use bursting for P20 and lower disks for workloads, such as batch jobs, workloads, which handle traffic spikes, and to improve OS boot time.
- ✓ Consider using Premium disks (P30 and greater).

Configuration recommendations

Consider the following recommendations to optimize costs when configuring your Azure managed disk:

Recommendation	Description
Configure data and log files on different disks for database workloads.	You can optimize IaaS DB workload performance by configuring system, data, and log files to be on different disk SKUs (leveraging Premium Disks for data and Ultra Disks for logs satisfies most production scenarios). Ultra Disk cost and performance can be optimized by taking advantage of configuring capacity, IOPS, and throughput independently. Also, you can dynamically configure these attributes. Example workloads include: - SQL on IaaS - Cassandra DB - Maria DB - MySql and - Mongo DB on IaaS

Recommendation	Description
Use bursting for P20 and lower disks for workloads, such as batch jobs, workloads, which handle traffic spikes, and to improve OS boot time.	Azure Disks offer various SKUs and sizes to satisfy different workload requirements. Some of the more recent features could help further optimize cost performance of existing disk use cases. You can use disk bursting for Premium (disks P20 and lower). Example scenarios that could benefit from this feature include: - Improving OS boot time - Handling batch jobs - Handling traffic spikes
Consider using Premium disks (P30 and greater).	Premium Disks (P30 and greater) can be reserved (one or three years) at a discounted price.

Next step

Event Grid and reliability

Event Grid and reliability

Article • 11/30/2022

Azure Event Grid lets you easily build applications with event-based architectures. This solution has build-in support for events coming from Azure services, like storage blobs and resource groups. Event Grid also has support for your own events, using custom topics.

For more information about using Event Grid, reference Create and route custom events with Azure Event Grid.

To understand how using Event Grid creates a more reliable workload, reference Server-side geo disaster recovery in Azure Event Grid.

The following sections are specific to Azure Event Grid and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Grid provides an uptime SLA. For more information, reference SLA for Event Grid.

Checklist

Have you configured Azure Event Grid with reliability in mind?

- ✓ Deploy an Event Grid instance per region, in case of a multi-region Azure solution.
- ✓ Monitor Event Grid for failed event delivery.
- ✓ Use batched events.
- ✓ Event batches can't exceed `1MB` in size.
- ✓ Configure and optimize batch-size selection during load testing.
- ✓ Ensure Event Grid messages are accepted with `HTTP` `200-204` responses only if delivering to an endpoint that holds custom code.
- ✓ Monitor Event Grid for failed event publishing.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Event Grid:

Recommendation	Description
Monitor Event Grid for failed event delivery.	The `Delivery Failed` metric will increase every time a message can't be delivered to an event handler (timeout or a non-`200-204 HTTP` status code). If an event can't be lost, set up a Dead-Letter-Queue (DLQ) storage account. A DLQ account is where events that can't be delivered after the maximum retry count will be placed. Optionally, implement a notification system on the DLQ storage account, for example, by handling a *new file* event through Event Grid.
Use batched events in high-throughput scenarios.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must be able to process these arrays.
Event batches can't exceed `1MB` in size.	If the message payload is large, only one or a few messages will fit in the batch. The consuming service will need to process more event batches. If your event has a large payload, consider storing it elsewhere, such as in blob storage, and passing a reference in the event. When integrating with third-party services through the CloudEvents schema, it's not recommended to exceed `64KB` events.
Configure and optimize batch-size selection during load testing.	Batch size selection depends on the payload size and the message volume.
Monitor Event Grid for failed event publishing.	The `Unmatched` metric will show messages that are published, but not matched to any subscription. Depending on your application architecture, the latter may be intentional.

Source artifacts

To determine the **Input Schema** type for all available Event Grid topics, use the following query:

```SQL
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid domains, use the following query:

```sql
Resources
| where type == 'microsoft.eventgrid/domains' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To identify **Public Network Access** status for all available Event Grid domains, use the following query:

```sql
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

To identify **Firewall Rules** for all public Event Grid domains, use the following query:

```sql
Resources
| where type == 'microsoft.eventgrid/domains' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To identify **Firewall Rules** for all public Event Grid topics, use the following query:

```sql
Resources
| where type == 'microsoft.eventgrid/topics' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To determine the **Input Schema** type for all available Event Grid domains, use the following schema:

SQL

```
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To identify **Public Network Access** status for all available Event Grid topics, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

Next step

Event Grid and operational excellence

Event Grid and operational excellence

Article • 11/30/2022

Azure Event Grid lets you easily build applications with event-based architectures. This solution has build-in support for events coming from Azure services, like storage blobs and resource groups. Event Grid also has support for your own events, using custom topics.

For more information about using Event Grid, reference Create and route custom events with Azure Event Grid.

To understand how using Event Grid promotes operational excellence for your workload, reference Diagnostic logs for Event Grid topics and Event Grid domains.

The following sections are specific to Azure Event Grid and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Grid provides an uptime SLA. For more information, reference SLA for Event Grid.

Checklist

Have you configured Azure Event Grid with operational excellence in mind?

- ✓ Monitor Event Grid for failed event delivery.
- ✓ Use batched events.
- ✓ Event batches can't exceed `1MB` in size.
- ✓ Configure and optimize batch-size selection during load testing.
- ✓ Ensure Event Grid messages are accepted with `HTTP` `200-204` responses only if delivering to an endpoint that holds custom code.
- ✓ Monitor Event Grid for failed event publishing.

Configuration recommendations

Consider the following recommendations to optimize operational excellence when configuring Azure Event Grid:

Recommendation	Description
Monitor Event Grid for failed event delivery.	The `Delivery Failed` metric will increase every time a message can't be delivered to an event handler (timeout or a non-`200-204 HTTP` status code). If an event can't be lost, set up a Dead-Letter-Queue (DLQ) storage account. A DLQ account is where events that can't be delivered after the maximum retry count will be placed. Optionally, implement a notification system on the DLQ storage account, for example, by handling a *new file* event through Event Grid.
Use batched events in high-throughput scenarios.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must be able to process these arrays.
Event batches can't exceed `1MB` in size.	If the message payload is large, only one or a few messages will fit in the batch. The consuming service will need to process more event batches. If your event has a large payload, consider storing it elsewhere, such as in blob storage, and passing a reference in the event. When integrating with third-party services through the CloudEvents schema, it's not recommended to exceed `64KB` events.
Configure and optimize batch-size selection during load testing.	Batch size selection depends on the payload size and the message volume.
Monitor Event Grid for failed event publishing.	The `Unmatched` metric will show messages that are published, but not matched to any subscription. Depending on your application architecture, the latter may be intentional.

Source artifacts

To determine the **Input Schema** type for all available Event Grid topics, use the following query:

SQL

```sql
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid domains, use the following query:

```SQL
Resources
| where type == 'microsoft.eventgrid/domains' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To identify **Public Network Access** status for all available Event Grid domains, use the following query:

```SQL
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

To identify **Firewall Rules** for all public Event Grid domains, use the following query:

```SQL
Resources
| where type == 'microsoft.eventgrid/domains' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To identify **Firewall Rules** for all public Event Grid topics, use the following query:

```SQL
Resources
| where type == 'microsoft.eventgrid/topics' and
properties['publicNetworkAccess'] == 'Enabled'
| project name, resourceGroup, location, subscriptionId,
properties['inboundIpRules']
```

To retrieve the **Resource ID** of existing private endpoints for Event Grid topics, use the following query:

```sql
Resources
| where type == 'microsoft.eventgrid/topics' and
notnull(properties['privateEndpointConnections'])
| mvexpand properties['privateEndpointConnections']
| project-rename privateEndpointConnections =
properties_privateEndpointConnections
| project name, resourceGroup, location, subscriptionId,
privateEndpointConnections['properties']['privateEndpoint']['id']
```

To determine the **Input Schema** type for all available Event Grid domains, use the following schema:

```sql
Resources
| where type == 'microsoft.eventgrid/domains'
| project name, resourceGroup, location, subscriptionId,
properties['inputSchema']
```

To identify **Public Network Access** status for all available Event Grid topics, use the following query:

```sql
Resources
| where type == 'microsoft.eventgrid/topics'
| project name, resourceGroup, location, subscriptionId,
properties['publicNetworkAccess']
```

Next step

Event Hubs and reliability

Event Hubs and reliability

Article • 11/30/2022

Azure Event Hubs is a scalable event processing service that ingests and processes large volumes of events and data, with low latency and high reliability. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching and storage adapters.

For more information about using Event Hubs, reference the Azure Event Hubs documentation to learn how to use Event Hubs to ingest millions of events per second from connected devices and applications.

To understand how using Event Hubs creates a more reliable workload, reference Azure Event Hubs - Geo-disaster recovery.

The following sections are specific to Azure Event Hubs and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Hubs provides an uptime SLA. For more information, reference SLA for Event Hubs.

Checklist

Have you configured Azure Event Hubs with reliability in mind?

- ✓ Create SendOnly and ListenOnly policies for the event publisher and consumer, respectively.
- ✓ When using the SDK to send events to Event Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.
- ✓ In high-throughput scenarios, use batched events.
- ✓ Every consumer can read events from one to `32` partitions.
- ✓ When developing new applications, use `EventProcessorClient` (.NET and Java) or `EventHubConsumerClient` (Python and JavaScript) as the client SDK.

- ✓ As part of your solution-wide availability and disaster recovery strategy, consider enabling the Event Hubs geo disaster-recovery option.
- ✓ When a solution has a large number of independent event publishers, consider using Event Publishers for fine-grained access control.
- ✓ Don't publish events to a specific partition.
- ✓ When publishing events frequently, use the AMQP protocol when possible.
- ✓ The number of partitions reflect the degree of downstream parallelism you can achieve.
- ✓ Ensure each consuming application uses a separate consumer group and only one active receiver per consumer group is in place.
- ✓ When using the Capture feature, carefully consider the configuration of the time window and file size, especially with low event volumes.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Event Hubs:

Recommendation	Description
When using the SDK to send events to Event Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.	When using `HTTPS`, ensure a proper retry pattern is implemented.
In high-throughput scenarios, use batched events.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must process these arrays.
Every consumer can read events from one to `32` partitions.	To achieve maximum scale on the side of the consuming application, every consumer should read from a single partition.
When developing new applications, use `EventProcessorClient` (.NET and Java) or `EventHubConsumerClient` (Python and JavaScript) as the client SDK.	`EventProcessorHost` has been deprecated.

Recommendation	Description
As part of your solution-wide availability and disaster recovery strategy, consider enabling the Event Hubs geo disaster-recovery option.	This option allows the creation of a secondary namespace in a different region. Only the active namespace receives messages at any time. Messages and events aren't replicated to the secondary region. The RTO for the regional failover is *up to 30 minutes*. Confirm this RTO aligns with the requirements of the customer and fits in the broader availability strategy. If a higher RTO is required, consider implementing a client-side failover pattern.
When a solution has a large number of independent event publishers, consider using Event Publishers for fine-grained access control.	Event Publishers automatically set the partition key to the publisher name, so this feature should only be used if the events originate from all publishers evenly.
Don't publish events to a specific partition.	If ordering events is essential, implement ordering downstream or use a different messaging service instead.
When publishing events frequently, use the AMQP protocol when possible.	AMQP has higher network costs when initializing the session, but `HTTPS` requires TLS overhead for every request. AMQP has higher performance for frequent publishers.
The number of partitions reflect the degree of downstream parallelism you can achieve.	For maximum throughput, use the maximum number of partitions (`32`) when creating the Event Hub. The maximum number of partitions will allow you to scale up to `32` concurrent processing entities and will offer the highest send and receive availability.
When using the Capture feature, carefully consider the configuration of the time window and file size, especially with low event volumes.	Data Lake will charge for minimal file size for storage (gen1) or minimal transaction size (gen2). If you set the time window so low that the file hasn't reached minimum size, you'll incur extra cost.

Source artifacts

To find Event Hubs namespaces with **Basic** SKU, use the following query:

SQL

```sql
Resources
| where type == 'microsoft.eventhub/namespaces'
| where sku.name == 'Basic'
| project resourceGroup, name, sku.name
```

Next step

Event Hubs and operational excellence

Event Hubs and operational excellence

Article • 11/30/2022

Azure Event Hubs is a scalable event processing service that ingests and processes large volumes of events and data, with low latency and high reliability. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching and storage adapters.

For more information about using Event Hubs, reference the Azure Event Hubs documentation to learn how to use Event Hubs to ingest millions of events per second from connected devices and applications.

To understand ways using Event Hubs helps you achieve operational excellence for your workload, reference the following articles:

- Monitor Azure Event Hubs
- Stream Azure Diagnostics data using Event Hubs
- Scaling with Event Hubs

The following sections are specific to Azure Event Hubs and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Event Hubs provides an uptime SLA. For more information, reference SLA for Event Hubs.

Checklist

Have you configured Azure Event Hubs with operational excellence in mind?

- ✓ Create SendOnly and ListenOnly policies for the event publisher and consumer, respectively.
- ✓ When using the SDK to send events to Event Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.
- ✓ In high-throughput scenarios, use batched events.

- ✓ Every consumer can read events from one to `32` partitions.
- ✓ When developing new applications, use `EventProcessorClient` (.NET and Java) or `EventHubConsumerClient` (Python and JavaScript) as the client SDK.
- ✓ As part of your solution-wide availability and disaster recovery strategy, consider enabling the Event Hubs geo disaster-recovery option.
- ✓ When a solution has a large number of independent event publishers, consider using Event Publishers for fine-grained access control.
- ✓ Don't publish events to a specific partition.
- ✓ When publishing events frequently, use the AMQP protocol when possible.
- ✓ The number of partitions reflect the degree of downstream parallelism you can achieve.
- ✓ Ensure each consuming application uses a separate consumer group and only one active receiver per consumer group is in place.
- ✓ When using the Capture feature, carefully consider the configuration of the time window and file size, especially with low event volumes.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Event Hubs:

Recommendation	Description
When using the SDK to send events to Event Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.	When using `HTTPS`, ensure a proper retry pattern is implemented.
In high-throughput scenarios, use batched events.	The service will deliver a `json` array with multiple events to the subscribers, instead of an array with one event. The consuming application must process these arrays.
Every consumer can read events from one to `32` partitions.	To achieve maximum scale on the side of the consuming application, every consumer should read from a single partition.

Recommendation	Description
When developing new applications, use `EventProcessorClient` (.NET and Java) or `EventHubConsumerClient` (Python and JavaScript) as the client SDK.	`EventProcessorHost` has been deprecated.
As part of your solution-wide availability and disaster recovery strategy, consider enabling the Event Hubs geo disaster-recovery option.	This option allows the creation of a secondary namespace in a different region. Only the active namespace receives messages at any time. Messages and events aren't replicated to the secondary region. The RTO for the regional failover is *up to 30 minutes*. Confirm this RTO aligns with the requirements of the customer and fits in the broader availability strategy. If a higher RTO is required, consider implementing a client-side failover pattern.
When a solution has a large number of independent event publishers, consider using Event Publishers for fine-grained access control.	Event Publishers automatically set the partition key to the publisher name, so this feature should only be used if the events originate from all publishers evenly.
Don't publish events to a specific partition.	If ordering events is essential, implement ordering downstream or use a different messaging service instead.
When publishing events frequently, use the AMQP protocol when possible.	AMQP has higher network costs when initializing the session, but `HTTPS` requires TLS overhead for every request. AMQP has higher performance for frequent publishers.
The number of partitions reflect the degree of downstream parallelism you can achieve.	For maximum throughput, use the maximum number of partitions (`32`) when creating the Event Hub. The maximum number of partitions will allow you to scale up to `32` concurrent processing entities and will offer the highest send and receive availability.
When using the Capture feature, carefully consider the configuration of the time window and file size, especially with low event volumes.	Data Lake will charge for minimal file size for storage (gen1) or minimal transaction size (gen2). If you set the time window so low that the file hasn't reached minimum size, you'll incur extra cost.

Source artifacts

To find Event Hubs namespaces with **Basic** SKU, use the following query:

SQL

```
Resources
| where type == 'microsoft.eventhub/namespaces'
| where sku.name == 'Basic'
| project resourceGroup, name, sku.name
```

Next step

Service Bus and reliability

Service Bus and reliability

Article • 11/30/2022

Fully manage enterprise message brokering with message queues and publish-subscribe topics used in Azure Service Bus. This service stores messages in a *broker* (for example, a *queue*) until the consuming party is ready to receive the messages.

Benefits include:

- Load-balancing across competing workers.
- Safely routing and transferring data and control across service, and application boundaries.
- Coordinating transactional work that requires a high-degree of reliability.

For more information about using Service Bus, reference Azure Service Bus Messaging. Learn how to set up messaging that connects applications and services across on-premises and cloud environments.

To understand how Service Bus contributes to a reliable workload, reference the following topics:

- Asynchronous messaging patterns and high availability
- Azure Service Bus Geo-disaster recovery
- Handling outages and disasters

The following sections are specific to Azure Service Bus and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Maximize reliability with an Azure Service Bus uptime SLA. Properly configured applications can send or receive messages, or do other operations on a deployed Queue or Topic. For more information, reference the Service Bus SLA.

Other design considerations include:

- Express Entities
- Partitioned queues and topics

Besides the documentation on Service Bus Premium and Standard messaging tiers, the following features are only available on the Premium Stock Keeping Unit (SKU):

- Dedicated resources.
- Virtual network integration: Limits the networks that can connect to the Service Bus instance. Requires Service Endpoints to be enabled on the subnet. There are Trusted Microsoft services that are not supported when implementing Virtual Networks(for example, integration with Event Grid). For more information, reference Allow access to Azure Service Bus namespace from specific virtual networks.
- Private endpoints.
- IP Filtering/Firewall: Restrict connections to only defined `IPv4` addresses or `IPv4` address ranges.
- Availability zones: Provides enhanced availability by spreading replicas across availability zones within one region at no extra cost.
- Event Grid integration: Available event types.
- Scale messaging units.
- Geo-Disaster Recovery (paired namespace).
- BYOK (Bring Your Own Key): Azure Service Bus encrypts data at rest and automatically decrypts it when accessed, but customers can also bring their own customer-managed key.

When deploying Service Bus with Geo-disaster recovery and in availability zones, the Service Level Operation (SLO) increases dramatically, but does not change the uptime SLA.

Checklist

Have you configured Azure Service Bus with reliability in mind?

- ✓ Evaluate Premier-tier benefits of Azure Service Bus.
- ✓ Ensure that Service Bus Messaging Exceptions are handled properly.
- ✓ Connect to Service Bus with the Advanced Messaging Queue Protocol (AMQP) and use Service Endpoints or Private Endpoints when possible.
- ✓ Review the Best Practices for performance improvements using Service Bus Messaging.
- ✓ Implement geo-replication on the sender and receiver side to protect against outages and disasters.
- ✓ Configure Geo-Disaster.
- ✓ If you need mission-critical messaging with queues and topics, Service Bus Premium is recommended with Geo-Disaster Recovery.

- ✓ Configure Zone Redundancy in the Service Bus namespace (*only available with Premium tier*).
- ✓ Implement high availability for the Service Bus namespace.
- ✓ Ensure related messages are delivered in guaranteed order.
- ✓ Evaluate different Java Messaging Service (JMS) features through the JMS API.
- ✓ Use .NET Nuget packages to communicate with Service Bus messaging entities.
- ✓ Implement resilience for transient fault handling when sending or receiving messages.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure Service Bus:

Recommendation	Description
Evaluate Premier-tier benefits of Azure Service Bus.	Consider migrating to the Premium tier of Service Bus to take advantage of platform-supported outage and disaster protection.
Connect to Service Bus with the AMQP protocol and use Service Endpoints or Private Endpoints when possible.	This recommendation keeps traffic on the Azure Backbone. *Note: The default connection protocol for `Microsoft.Azure.ServiceBus` and `Windows.Azure.ServiceBus` namespaces is `AMQP`*.
Implement geo-replication on the sender and receiver side to protect against outages and disasters.	Standard tier supports only the implementation of sender and receiver-side geo-redundancy. An outage or disaster in an Azure Region could cause downtime for your solution.
Configure Geo-Disaster.	- Active/Active - Active/Passive - Paired Namespace (Active/Passive) - *Note: The secondary region should preferably be an Azure paired region*.
If you need mission-critical messaging with queues and topics, Service Bus Premium is recommended with Geo-Disaster Recovery.	Choosing the pattern is dependent on the business requirements and the recovery time objective (RTO).

Recommendation	Description
Configure Zone Redundancy in the Service Bus namespace (*only available with Premium tier*).	Zone Redundancy includes three copies of the messaging store. One zone is allocated as the primary messaging store and the other zones are allocated as secondaries. If the primary zone becomes unavailable, a secondary is promoted to primary with no perceivable downtime. *Availability Zones are available in a subset of Azure Regions with new regions added regularly.*
Implement high availability for the Service Bus namespace.	Premium tier supports Geo-disaster recovery and replication at the namespace level. At this level, Premium tier provides high availability for metadata disaster recovery using primary and secondary disaster recovery namespaces.
Ensure related messages are delivered in guaranteed order.	Be aware of the requirement to set a Partition Key, Session ID, or Message ID on each message to ensure related messages send to the same partition in the messaging entity.
Evaluate different JMS features through the JMS API.	Features available through the JMS 2.0 API (and its Software Development Kit (SDK)) are not the same as the features available through the native SDK. For example, Service Bus Sessions are not available in JMS.
Implement resilience for transient fault handling when sending or receiving messages.	It is essential to implement suitable transient fault handling and error handling for send and receive operations to maintain throughput and to prevent message loss.

Source artifacts

- To identify premium Service Bus Instances that are not using private endpoints, use the following query:

  SQL

  ```
  Resources
  | where
    type == 'microsoft.servicebus/namespaces'
  | where
    sku.tier == 'Premium'
    and isempty(properties.privateEndpointConnections)
  ```

- To identify Service Bus Instances that are not on the premium tier, use the following query:

  SQL

```
Resources
| where
  type == 'microsoft.servicebus/namespaces'
| where
  sku.tier != 'Premium'
```

- To identify premium Service Bus Instances that are not zone redundant, use the following query:

SQL

```sql
Resources
| where
  type == 'microsoft.servicebus/namespaces'
| where
  sku.tier == 'Premium'
  and properties.zoneRedundant == 'false'
```

Next step

Service Bus and operational excellence

Service Bus and operational excellence

Article • 11/30/2022

Fully manage enterprise message brokering with message queues and publish-subscribe topics using Azure Service Bus. This service stores messages in a *broker* (for example, a *queue*) until the consuming party is ready to receive the messages.

Benefits include:

- Load-balancing work across competing workers.
- Safely routing and transferring data and control across service, and application boundaries.
- Coordinating transactional work that requires a high-degree of reliability.

For more information about using Service Bus, reference Azure Service Bus Messaging. Learn how to set up messaging that connects applications and services across on-premises and cloud environments.

To understand how Service Bus promotes operational excellence, reference the following topics:

- Handling outages and disasters
- Throttling operations on Azure Service Bus

The following sections are specific to Azure Service Bus and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Maximize reliability with an Azure Service Bus uptime Service Level Agreement (SLA). Properly configured applications can send or receive messages, or do other operations on a deployed Queue or Topic. For more information, reference the Service Bus SLA.

Other design considerations include:

- Express Entities
- Partitioned queues and topics

Besides the documentation on Service Bus Premium and Standard messaging tiers, the following features are only available on the Premium Stock Keeping Unit (SKU):

- Dedicated resources.
- Virtual network integration: Limits the networks that can connect to the Service Bus instance. Requires Service Endpoints to be enabled on the subnet. There are Trusted Microsoft services that are not supported when implementing Virtual Networks (for example, integration with Event Grid). For more information, reference Allow access to Azure Service Bus namespace from specific virtual networks.
- Private endpoints.
- IP Filtering/Firewall: Restrict connections to only defined `IPv4` addresses or `IPv4` address ranges.
- Availability zones: Provides enhanced availability by spreading replicas across availability zones within one region at no extra cost.
- Event Grid integration: Available event types.
- Scale messaging units.
- Geo-Disaster Recovery (paired namespace).
- BYOK (Bring Your Own Key): Azure Service Bus encrypts data at rest and automatically decrypts it when accessed, but customers can also bring their own customer-managed key.

When deploying Service Bus with Geo-disaster recovery and in availability zones, the Service Level Objective (SLO) increases dramatically, but does not change the uptime SLA.

Checklist

Have you configured Azure Service Bus with operational excellence in mind?

- ✓ Ensure that Service Bus Messaging Exceptions are handled properly.
- ✓ Connect to Service Bus with the Advanced Message Queuing Protocol (AMQP) and use Service Endpoints or Private Endpoints when possible.
- ✓ Establish a process to actively monitor the dead-letter queue (dlq) messages.
- ✓ Review the Best Practices for performance improvements using Service Bus Messaging.
- ✓ Analyze the differences between Azure Storage Queues and Azure Service Bus Queues.

Configuration recommendations

Consider the following recommendation to optimize reliability when configuring Azure Service Bus:

Recommendation	Description
Connect to Service Bus with the AMQP protocol and use Service Endpoints or Private Endpoints when possible.	This recommendation keeps traffic on the Azure Backbone. *Note: The default connection protocol for* `Microsoft.Azure.ServiceBus` *and* `Windows.Azure.ServiceBus` *namespaces is* `AMQP`.
Establish a process to actively monitor the dead-letter queue (dlq) messages.	The dead-letter queue holds messages that cannot be processed or cannot be delivered to any receiver. It is important to monitor this queue to examine the issue cause, apply required corrections, and to resubmit messages.
Analyze the differences between Azure Storage Queues and Azure Service Bus Queues.	You will find that Azure Service Bus Messaging Entities are more advanced, reliable, and feature-rich than Azure Storage Queues. If your requirement is for simple queue messaging without requirements for reliable messaging, then Azure Storage Queues may be a more suitable option.

Source artifacts

- To identify premium Service Bus Instances that aren't using private endpoints, use the following query:

  SQL

  ```
  Resources
  | where
    type == 'microsoft.servicebus/namespaces'
  | where
    sku.tier == 'Premium'
    and isempty(properties.privateEndpointConnections)
  ```

- To identify Service Bus Instances that are not on the premium tier, use the following query:

  SQL

  ```
  Resources
  | where
    type == 'microsoft.servicebus/namespaces'
  | where
    sku.tier != 'Premium'
  ```

Next step

Queue Storage and reliability

Queue Storage and reliability

Article • 11/30/2022

Azure Queue Storage is a service for storing large numbers of messages that you can access from anywhere in the world through authenticated calls using `HTTP` or `HTTPS`. Queues are commonly used to create a backlog of work to process asynchronously.

For more information about Queue Storage, reference What is Azure Queue Storage?

To understand how Azure Queue Storage helps maintain a reliable workload, reference the following topics:

- Azure Storage redundancy
- Disaster recovery and storage account failover

The following sections are specific to Azure Queue Storage and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options
- Source artifacts

Design considerations

Azure Queue Storage follows the SLA statements of the general Storage Account service.

Checklist

Have you configured Azure Queue Storage with reliability in mind?

- ✓ Since Storage Queues are a part of the Azure Storage service, refer to the Storage Accounts configuration checklist and recommendations for reliability.
- ✓ Ensure that for all clients accessing the storage account, implement a proper retry policy.
- ✓ Refer to the Storage guidance for specifics on data recovery for storage accounts.
- ✓ For an SLA increase, use geo-redundant storage.
- ✓ Use geo-zone-redundant storage (GZRS) or read-access geo-zone-redundant storage (RA-GZRS) for durability and protection against failover if an entire data center becomes unavailable.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your Azure Queue Storage:

Recommendation	Description
For an SLA increase, use geo-redundant storage.	Use geo-redundant storage with read access and configure the client application to fail over to secondary read endpoints if the primary endpoints fail to respond. This consideration should be part of the overall reliability strategy of your solution.
Use geo-zone-redundant storage (GZRS) or read-access geo-zone-redundant storage (RA-GZRS) for durability and protection against failover if an entire data center becomes unavailable.	For more information, reference Azure Storage redundancy.

Source artifacts

To identify storage accounts using locally redundant storage (LRS), use the following query:

```SQL
Resources
| where
    type == 'microsoft.storage/storageaccounts'
    and sku.name =~ 'Standard_LRS'
```

To identify storage accounts using V1 storage accounts, use the following query:

```SQL
Resources
| where
    type == 'microsoft.storage/storageaccounts'
    and kind == 'Storage'
```

Next step

Queue Storage and operational excellence

Queue Storage and operational excellence

Article • 11/30/2022

Azure Queue Storage is a service for storing large numbers of messages that you can access from anywhere in the world through authenticated calls using `HTTP` or `HTTPS`. Queues are commonly used to create a backlog of work to process asynchronously.

For more information about Queue Storage, reference What is Azure Queue Storage?

To understand how Azure Queue Storage promotes operational excellence, reference the following topics:

- Monitoring Azure Queue Storage
- Best practices for monitoring Azure Queue Storage

The following sections are specific to Azure Queue Storage and operational excellence:

- Design considerations
- Configuration checklist
- Source artifacts

Design considerations

Azure Queue Storage follows the SLA statements of the general Storage Account service.

Checklist

Have you configured Azure Queue Storage with operational excellence in mind?

- ✓ Since Storage Queues are a part of the Azure Storage service, refer to the Storage Accounts configuration checklist and recommendations for operational excellence.
- ✓ Ensure that for all clients accessing the storage account, implement a proper retry policy.
- ✓ Refer to the Storage guidance for specifics on data recovery for storage accounts.

Source artifacts

To identify storage accounts using V1 storage accounts, use the following query:

SQL

```sql
Resources
| where
    type == 'microsoft.storage/storageaccounts'
    and kind == 'Storage'
```

Next step

IoT Hub and reliability

IoT Hub and reliability

Article • 11/30/2022

Azure IoT Hub is a managed service hosted in the cloud that acts as a central message hub for communication between an IoT application and its attached devices. You can connect millions of devices and their backend solutions reliably and securely. Almost any device can be connected to an IoT Hub.

IoT Hub supports monitoring to help you track device creation, device connections, and device failures.

IoT Hub also supports the following messaging patterns:

- Device-to-cloud telemetry
- Uploading files from devices
- Request-reply methods to control your devices from the cloud

For more information about IoT Hub, reference IoT Concepts and Azure IoT Hub.

To understand how IoT Hub supports a reliable workload, reference the following topics:

- IoT Hub high availability and disaster recovery
- How to achieve cross-region High Availability with IoT Hub
- How to clone an Azure IoT Hub to another region

The following sections are specific to Azure IoT Hub and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options

Design considerations

For more information about the Azure IoT Hub Service Level Agreement, reference SLA for Azure IoT Hub .

Checklist

Have you configured Azure IoT Hub with reliability in mind?

- ✓ Provision a second IoT Hub in another region and have routing logic on the device.
- ✓ Use the `AMQP` or `MQTT` protocol when sending events frequently.

- ✓ Use only certificates validated by a root CA in the production environment if you're using X.509 certificates for the device connection.
- ✓ For maximum throughput, use the maximum number of partitions (`32`) when creating the IoT Hub, if you're planning to use the built-in endpoint.
- ✓ For scaling, increase the tier and allocated IoT Hub units instead of adding more than one IoT Hub per region.
- ✓ In high-throughput scenarios, use batched events.
- ✓ If you require the minimum possible latency, don't use routing and read the events from the built-in endpoint.
- ✓ As part of your solution-wide availability and disaster recovery strategy, consider using the IoT Hub cross-region Disaster Recovery option.
- ✓ When reading device telemetry from the built-in Event Hub-compatible endpoint, refer to the Event Hub consumers recommendation.
- ✓ When using an SDK to send events to IoT Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.
- ✓ To avoid telemetry interruption due to throttling and a fully used quota, consider adding a custom auto-scaling solution.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Azure IoT Hub:

Recommendation	Description
Provision a second IoT Hub in another region and have routing logic on the device.	These configurations can be further enhanced with a Concierge Service.
Use the `AMQP` or `MQTT` protocol when sending events frequently.	`AMQP` and `MQTT` have higher network costs when initializing the session, however `HTTPS` requires extra TLS overhead for every request. `AMQP` and `MQTT` have higher performance for frequent publishers.
Use only certificates validated by a root CA in the production environment if you're using X.509 certificates for the device connection.	Make sure you have processes in place to update the certificate before they expire.

Recommendation	Description
For maximum throughput, use the maximum number of partitions (`32`) when creating the IoT Hub, if you're planning to use the built-in endpoint.	The number of device-to-cloud partitions for the Event Hub-compatible endpoint reflect the degree of downstream parallelism you can achieve. This will allow you to scale up to `32` concurrent processing entities and will offer the highest send and receive availability. This number can't be changed after creation.
For scaling, increase the tier and allocated IoT Hub units instead of adding more than one IoT Hub per region.	Adding more than one IoT Hub per region doesn't offer extra resiliency because all hubs can run on the same underlying cluster.
In high-throughput scenarios, use batched events.	The service will deliver an array with multiple events to the consumers, instead of an array with one event. The consuming application must process these arrays.
If you require the minimum possible latency, don't use routing and read the events from the built-in endpoint.	When using message routing in IoT Hub, latency of the message delivery increases. On average, latency shouldn't exceed `500 ms`, but there's no guarantee for the delivery latency.
As part of your solution-wide availability and disaster recovery strategy, consider using the IoT Hub cross-region Disaster Recovery option.	This option will move the IoT Hub endpoint to the paired Azure region. Only the device registry gets replicated. Events aren't replicated to the secondary region. *The RTO for the customer-initiated failover is between 10 minutes to a couple of hours. For a Microsoft-initiated failover, the RTO is `2-26` hours. Confirm this RTO aligns with the requirements of the customer and fits in the broader availability strategy. If a higher RTO is required, consider implementing a client-side failover pattern.*
When using an SDK to send events to IoT Hub, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.	When using `HTTPS`, implement a proper retry pattern.

Next step

IoT Hub and operational excellence

IoT Hub and operational excellence

Article • 11/30/2022

Azure IoT Hub is a managed service hosted in the cloud that acts as a central message hub for communication between an IoT application and its attached devices. You can connect millions of devices and their backend solutions reliably and securely. Almost any device can be connected to an IoT Hub.

IoT Hub supports monitoring to help you track device creation, device connections, and device failures.

IoT Hub also supports the following messaging patterns:

- Device-to-cloud telemetry
- Uploading files from devices
- Request-reply methods to control your devices from the cloud

For more information about IoT Hub, reference IoT Concepts and Azure IoT Hub.

To understand how IoT Hub promotes operational excellence, reference the following topics:

- Tutorial: Set up and use metrics and logs with an IoT Hub
- Monitoring Azure IoT Hub
- Trace Azure IoT device-to-cloud messages with distributed tracing (preview)
- Check IoT Hub service and resource health

The following sections are specific to Azure IoT Hub and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options

Design considerations

For more information about the Azure IoT Hub Service Level Agreement, reference SLA for Azure IoT Hub.

Checklist

Have you configured Azure IoT Hub with operational excellence in mind?

- ✓ Provision a second IoT Hub in another region and have routing logic on the device.
- ✓ Use the `AMQP` or `MQTT` protocol when sending events frequently.
- ✓ Use only certificates validated by a root CA in the production environment if you're using X.509 certificates for the device connection.
- ✓ For maximum throughput, use the maximum number of partitions (`32`) when creating the IoT Hub, if you're planning to use the built-in endpoint.
- ✓ For scaling, increase the tier and allocated IoT Hub units instead of adding more than one IoT Hub per region.
- ✓ In high-throughput scenarios, use batched events.
- ✓ If you require the minimum possible latency, don't use routing and read the events from the built-in endpoint.
- ✓ As part of your solution-wide availability and disaster recovery strategy, consider using the IoT Hub cross-region Disaster Recovery option.
- ✓ When reading device telemetry from the built-in Event Hub-compatible endpoint, refer to the Event Hub consumers recommendation.
- ✓ When using an SDK to send events to IoT Hubs, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.
- ✓ To avoid telemetry interruption due to throttling and a fully used quota, consider adding a custom auto-scaling solution.

Configuration recommendations

Consider the following recommendations for increasing operational excellence when configuring Azure IoT Hub:

Recommendation	Description
Provision a second IoT Hub in another region and have routing logic on the device.	These configurations can be further enhanced with a Concierge Service.
Use the `AMQP` or `MQTT` protocol when sending events frequently.	`AMQP` and `MQTT` have higher network costs when initializing the session, however `HTTPS` requires extra TLS overhead for every request. `AMQP` and `MQTT` have higher performance for frequent publishers.
Use only certificates validated by a root CA in the production environment if you're using X.509 certificates for the device connection.	Make sure you have processes in place to update the certificate before they expire.

Recommendation	Description
For maximum throughput, use the maximum number of partitions (`32`) when creating the IoT Hub, if you're planning to use the built-in endpoint.	The number of device-to-cloud partitions for the Event Hub-compatible endpoint reflect the degree of downstream parallelism you can achieve. This will allow you to scale up to `32` concurrent processing entities and will offer the highest send and receive availability. This number can't be changed after creation.
For scaling, increase the tier and allocated IoT Hub units instead of adding more than one IoT Hub per region.	Adding more than one IoT Hub per region doesn't offer extra resiliency because all hubs can run on the same underlying cluster.
In high-throughput scenarios, use batched events.	The service will deliver an array with multiple events to the consumers, instead of an array with one event. The consuming application must process these arrays.
If you require the minimum possible latency, don't use routing and read the events from the built-in endpoint.	When using message routing in IoT Hub, latency of the message delivery increases. On average, latency shouldn't exceed `500 ms`, but there's no guarantee for the delivery latency.
As part of your solution-wide availability and disaster recovery strategy, consider using the IoT Hub cross-region Disaster Recovery option.	This option will move the IoT Hub endpoint to the paired Azure region. Only the device registry gets replicated. Events aren't replicated to the secondary region. *The RTO for the customer-initiated failover is between 10 minutes to a couple of hours. For a Microsoft-initiated failover, the RTO is `2-26` hours. Confirm this RTO aligns with the requirements of the customer and fits in the broader availability strategy. If a higher RTO is required, consider implementing a client-side failover pattern.*
When using an SDK to send events to IoT Hub, ensure the exceptions thrown by the retry policy (`EventHubsException` or `OperationCancelledException`) are properly caught.	When using `HTTPS`, implement a proper retry pattern.

Next step

IoT Hub Device Provisioning Service and reliability

IoT Hub Device Provisioning Service and reliability

Article • 11/30/2022

The IoT Hub Device Provisioning Service (DPS) is a helper service for IoT Hub. DPS enables zero-touch, just-in-time provisioning to the right IoT Hub without requiring human intervention. IoT Hub DPS allows customers to provision millions of devices in a secure and scalable manner.

For more information, reference What is Azure IoT Hub Device Provisioning Service?

To understand how IoT Hub DPS can increase workload reliability, reference IoT Hub DPS supports Availability Zones.

Design considerations

For more information about the Service Level Agreement for Azure IoT Hub DPS, reference SLA for Azure IoT Hub .

Next step

IoT Hub Device Provisioning Service and operational excellence

IoT Hub Device Provisioning Service and operational excellence

Article • 11/30/2022

The IoT Hub Device Provisioning Service (DPS) is a helper service for IoT Hub. DPS enables zero-touch, just-in-time provisioning to the right IoT Hub without requiring human intervention. IoT Hub DPS allows customers to provision millions of devices in a secure and scalable manner.

For more information, reference What is Azure IoT Hub Device Provisioning Service?

To understand how IoT Hub DPS promotes operational excellence, reference How to manage device enrollments with Azure portal.

Design considerations

For more information about the Service Level Agreement for Azure IoT Hub DPS, reference SLA for Azure IoT Hub .

Next step

Application Delivery (General) and reliability

Application Delivery (General) and reliability

Article • 02/28/2023

Application Delivery (General) explores key recommendations to deliver internal-facing and external-facing applications in the Azure network through a secure, highly scalable, and highly available way. General Application Delivery can be handled using a combination of the following networking services in Azure:

- Content Delivery Network (CDN)
- Azure Front Door Service
- Traffic Manager
- Application Gateway
- Internet Analyzer
- Load Balancer

For more information, reference Azure networking services overview.

The following sections are specific to general Application Delivery and reliability:

- Design considerations
- Configuration checklist
- Recommended configuration options

Design considerations

Application Delivery in Azure includes the following design considerations:

- Azure Load Balancer (internal and public) provides high availability for application delivery at either a regional or global level. (*Standard tier only*)
- Azure Traffic manager allows the delivery of applications through DNS redirection, including traffic using protocols other than `HTTP/S`.
- Azure Front Door allows the secure delivery of highly available `HTTP/S` applications across Azure regions.
- Azure Application Gateway allows the secure delivery of `HTTP/S` applications at a regional level.

Checklist

Have you configured your Application Delivery networking services with reliability in mind?

- ✓ Use Azure Traffic Manager to deliver global applications that span protocols other than `HTTP/S`.
- ✓ When using Azure Front Door and Application Gateway to protect `HTTP/S` applications, use Web Application Firewall (WAF) policies in Front Door and lock down Application Gateway to receive traffic only from Azure Front Door.
- ✓ Create a separate health endpoint on the backend that the health probe can use. The health endpoint can aggregate the state of the critical services and dependencies needed to serve requests.
- ✓ Enable health probes for backends.
- ✓ Deploy Application Gateway v2 or third-party NVAs used for inbound `HTTP/S` connections together with the applications that they're securing.
- ✓ When doing global load balancing for `HTTP/S` applications, use Front Door over Traffic Manager.
- ✓ Use a third-party Network Virtual Appliance (NVA) if Application Gateway v2 can't be used for the security of `HTTP/S` applications.
- ✓ For secure delivery of `HTTP/S` applications, ensure you enable Web Application Firewall (WAF) protection and policies.
- ✓ Application delivery for both internal and external facing applications should be part of the application.
- ✓ Global `HTTP/S` applications that span Azure regions should be delivered and protected using Azure Front Door with Web Application Firewall (WAF) policies.
- ✓ All public IP addresses in the solution should be protected with a DDoS Standard protection plan.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your Application Delivery networking services:

Recommendation	Description
Use Azure Traffic Manager to deliver global applications that span protocols other than `HTTP/S`.	Traffic manager doesn't forward traffic, but only completes the DNS redirection. The connection from the client is established directly to the target using any protocol.

Recommendation	Description
When using Azure Front Door and Application Gateway to protect `HTTP/S` applications, use WAF policies in Front Door and lock down Application Gateway to receive traffic only from Azure Front Door.	Certain scenarios might force a customer to implement rules specifically on AppGateway: For example, if ModSec CRS `2.2.9`, CRS `3.0` or CRS `3.1` rules are required, rules can be only implemented on AppGateway. Conversely, rate-limiting and geo-filtering are available only on Azure Front Door, not on AppGateway. Instructions on how to lock down traffic can be found at FAQ for Azure Front Door.
Create a separate health endpoint on the backend that the health probe can use. The health endpoint can aggregate the state of the critical services and dependencies needed to serve requests.	For more information, reference Health Endpoint Monitoring pattern.
Enable health probes for backends.	Health probes are `http(s)` endpoints that are queried by the load balancer (Azure Front Door, Traffic Manager, AppGateway) service to determine if the backend is healthy enough to handle requests.
Deploy Application Gateway v2 or third-party NVAs used for inbound `HTTP/S` connections together with the applications that they're securing.	Don't centrally manage Application Gateway v2 or third-party NVAs within the organization and share with other workloads.
When doing global load balancing for `HTTP/S` applications, use Front Door over Traffic Manager.	- Azure Front Door optimizes the number of TCP connections to the backend when forwarding traffic. - Changes to the routing configuration, based on backend health, are instantaneous. With Traffic Manager, traffic will point to the original backend until a new DNS lookup occurs, plus potential time for DNS propagation. - Front Door supports caching on global edge nodes, negating the need for a separate CDN service. - Front Door supports Web Application Firewall rules, negating the need for a separate WAF service.
For secure delivery of `HTTP/S` applications, ensure you enable Web Application Firewall (WAF) protection and policies.	Enable WAF protection and policies in either Application Gateway or Front Door.

Recommendation	Description
Application delivery for both internal and external facing applications should be part of the application.	Don't centrally manage Application Delivery within an organization.

Next step

Application Delivery (General) and operational excellence

Application Delivery (General) and operational excellence

Article • 11/30/2022

Application Delivery (General) explores key recommendations to deliver internal-facing and external-facing applications in the Azure network through a secure, highly scalable, and highly available way. General Application Delivery can be handled using a combination of the following networking services in Azure:

- Content Delivery Network (CDN)
- Azure Front Door Service
- Traffic Manager
- Application Gateway
- Internet Analyzer
- Load Balancer

For more information, reference Azure networking services overview.

The following sections are specific to general Application Delivery and operational excellence:

- Design considerations
- Configuration checklist
- Recommended configuration options

Design considerations

Application Delivery in Azure includes the following design considerations:

- Azure Load Balancer (internal and public) provides high availability for application delivery at a regional level. (*Standard tier only*)
- Azure Traffic manager allows the delivery of applications through DNS redirection, including traffic using protocols other than `HTTP/S`.
- Azure Front Door allows the secure delivery of highly available `HTTP/S` applications across Azure regions.
- Azure Application Gateway allows the secure delivery of `HTTP/S` applications at a regional level.

Checklist

Have you configured your Application Delivery networking services with operational excellence in mind?

- ✓ Use Azure Traffic Manager to deliver global applications that span protocols other than `HTTP/S`.
- ✓ When using Azure Front Door and Application Gateway to protect `HTTP/S` applications, use Web Application Firewall (WAF) policies in Front Door and lock down Application Gateway to receive traffic only from Azure Front Door.
- ✓ Create a separate health endpoint on the backend that the health probe can use. The health endpoint can aggregate the state of the critical services and dependencies needed to serve requests.
- ✓ Enable health probes for backends.
- ✓ Deploy Application Gateway v2 or third-party NVAs used for inbound `HTTP/S` connections together with the applications that they're securing.
- ✓ When doing global load balancing for `HTTP/S` applications, use Front Door over Traffic Manager.
- ✓ Use a third-party Network Virtual Appliance (NVA) if Application Gateway v2 can't be used for the security of `HTTP/S` applications.
- ✓ For secure delivery of `HTTP/S` applications, ensure you enable Web Application Firewall (WAF) protection and policies.
- ✓ Application delivery for both internal and external facing applications should be part of the application.
- ✓ Global `HTTP/S` applications that span Azure regions should be delivered and protected using Azure Front Door with Web Application Firewall (WAF) policies.
- ✓ All public IP addresses in the solution should be protected with a DDoS Standard protection plan.

Configuration recommendations

Consider the following recommendations for operational excellence when configuring your Application Delivery networking services:

Recommendation	Description
Use Azure Traffic Manager to deliver global applications that span protocols other than `HTTP/S`.	Traffic manager doesn't forward traffic, but only completes the DNS redirection. The connection from the client is established directly to the target using any protocol.

Recommendation	Description
When using Azure Front Door and Application Gateway to protect `HTTP/S` applications, use WAF policies in Front Door and lock down Application Gateway to receive traffic only from Azure Front Door.	Certain scenarios might force a customer to implement rules specifically on AppGateway: For example, if ModSec CRS `2.2.9`, CRS `3.0` or CRS `3.1` rules are required, rules can be only implemented on AppGateway. Conversely, rate-limiting and geo-filtering are available only on Azure Front Door, not on AppGateway. Instructions on how to lock down traffic can be found at FAQ for Azure Front Door.
Create a separate health endpoint on the backend that the health probe can use. The health endpoint can aggregate the state of the critical services and dependencies needed to serve requests.	For more information, reference Health Endpoint Monitoring pattern.
Enable health probes for backends.	Health probes are `http(s)` endpoints that are queried by the load balancer (Azure Front Door, Traffic Manager, AppGateway) service to determine if the backend is healthy enough to handle requests.
Deploy Application Gateway v2 or third-party NVAs used for inbound `HTTP/S` connections together with the applications that they're securing.	Don't centrally manage Application Gateway v2 or third-party NVAs within the organization and share with other workloads.
When doing global load balancing for `HTTP/S` applications, use Front Door over Traffic Manager.	- Azure Front Door optimizes the number of TCP connections to the backend when forwarding traffic. - Changes to the routing configuration, based on backend health, are instantaneous. With Traffic Manager, traffic will point to the original backend until a new DNS lookup occurs, plus potential time for DNS propagation. - Front Door supports caching on global edge nodes, negating the need for a separate CDN service. - Front Door supports Web Application Firewall rules, negating the need for a separate WAF service.
For secure delivery of `HTTP/S` applications, ensure you enable Web Application Firewall (WAF) protection and policies.	Enable WAF protection and policies in either Application Gateway or Front Door.

Recommendation	Description
Application delivery for both internal and external facing applications should be part of the application.	Don't centrally manage Application Delivery within an organization.

Next step

Azure Well-Architected Framework review - Azure Application Gateway v2

Azure Well-Architected Framework review - Azure Application Gateway v2

Article • 03/20/2023

This article provides architectural best practices for the Azure Application Gateway v2 family of SKUs. The guidance is based on the five pillars of architecture excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

We assume that you have working knowledge of Azure Application Gateway and are well versed with v2 SKU features. For more information, see Azure Application Gateway features.

Prerequisites

- Understanding the Well-Architected Framework pillars can help produce a high-quality, stable, and efficient cloud architecture. We recommend that you review your workload by using the Azure Well-Architected Framework Review assessment.
- Use a reference architecture to review the considerations based on the guidance provided in this article. We recommend that you start with Protect APIs with Application Gateway and API Management and IaaS: Web application with relational database.

Reliability

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Use the following information to minimize failed instances.

Design checklist

As you make design choices for Application Gateway, review the Reliability design principles.

✓ Deploy the instances in a zone-aware configuration, where available.

- ✓ Use Application Gateway with Web Application Firewall (WAF) within an application virtual network to protect inbound `HTTP/S` traffic from the Internet.
- ✓ In new deployments, use Application Gateway v2 unless there is a compelling reason to use v1.
- ✓ Plan for rule updates
- ✓ Use health probes to detect backend unavailability
- ✓ Review the impact of the interval and threshold settings on health probes
- ✓ Verify downstream dependencies through health endpoints

Recommendations

Explore the following table of recommendations to optimize your Application Gateway configuration for Reliability.

Recommendation	Benefit
Plan for rule updates	Plan enough time for updates before accessing Application Gateway or making further changes. For example, removing servers from backend pool might take some time because they have to drain existing connections.
Use health probes to detect backend unavailability	If Application Gateway is used to load balance incoming traffic over multiple backend instances, we recommend the use of health probes. These will ensure that traffic is not routed to backends that are unable to handle the traffic.
Review the impact of the interval and threshold settings on health probes	The health probe sends requests to the configured endpoint at a set interval. Also, there's a threshold of failed requests that will be tolerated before the backend is marked unhealthy. These numbers present a trade-off. - Setting a higher interval puts a higher load on your service. Each Application Gateway instance sends its own health probes, so 100 instances every 30 seconds means 100 requests per 30 seconds. - Setting a lower interval leaves more time before an outage is detected. - Setting a low unhealthy threshold may mean that short, transient failures may take down a backend. - Setting a high threshold it can take longer to take a backend out of rotation.

Recommendation	Benefit
Verify downstream dependencies through health endpoints	Suppose each backend has its own dependencies to ensure failures are isolated. For example, an application hosted behind Application Gateway may have multiple backends, each connected to a different database (replica). When such a dependency fails, the application may be working but won't return valid results. For that reason, the health endpoint should ideally validate all dependencies. Keep in mind that if each call to the health endpoint has a direct dependency call, that database would receive 100 queries every 30 seconds instead of 1. To avoid this, the health endpoint should cache the state of the dependencies for a short period of time.
When using Azure Front Door and Application Gateway to protect `HTTP/S` applications, use WAF policies in Front Door and lock down Application Gateway to receive traffic only from Azure Front Door.	Certain scenarios can force you to implement rules specifically on Application Gateway. For example, if ModSec CRS 2.2.9, CRS 3.0 or CRS 3.1 rules are required, these rules can be only implemented on Application Gateway. Conversely, rate-limiting and geo-filtering are available only on Azure Front Door, not on AppGateway.

Azure Advisor helps you ensure and improve continuity of your business-critical applications. Review the Azure Advisor recommendations.

Security

Security is one of the most important aspects of any architecture. Application Gateway provides features to employ both the principle of least privilege and defense-in-defense. We recommend you review the Security design principles.

Design checklist

- ✓ Set up a TLS policy for enhanced security
- ✓ Use AppGateway for TLS termination
- ✓ Use Azure Key Vault to store TLS certificates
- ✓ When re-encrypting backend traffic, ensure the backend server certificate contains both the root and intermediate Certificate Authorities (CAs)
- ✓ Use an appropriate DNS server for backend pool resources
- ✓ Comply with all NSG restrictions for Application Gateway
- ✓ Refrain from using UDRs on the Application Gateway subnet

✓ Be aware of Application Gateway capacity changes when enabling WAF

Recommendations

Explore the following table of recommendations to optimize your Application Gateway configuration for Security.

Recommendation	Benefit
Set up a TLS policy for enhanced security	Set up a TLS policy for extra security. Ensure you're using the latest TLS policy version (AppGwSslPolicy20170401S). This enforces TLS 1.2 and stronger ciphers.
Use AppGateway for TLS termination	There are advantages of using Application Gateway for TLS termination: - Performance improves because requests going to different backends to have to re-authenticate to each backend. - Better utilization of backend servers because they don't have to perform TLS processing - Intelligent routing by accessing the request content. - Easier certificate management because the certificate only needs to be installed on Application Gateway.
Use Azure Key Vault to store TLS certificates	Application Gateway is integrated with Key Vault. This provides stronger security, easier separation of roles and responsibilities, support for managed certificates, and an easier certificate renewal and rotation process.
When re-encrypting backend traffic, ensure the backend server certificate contains both the root and intermediate Certificate Authorities (CAs)	A TLS certificate of the backend server must be issued by a well-known CA. If the certificate was not issued by a trusted CA, the Application Gateway checks if the certificate of the issuing CA was issued by a trusted CA, and so on until either a trusted CA is found. Only then a secure connection is established. Otherwise, Application Gateway marks the backend as unhealthy.
Use an appropriate DNS server for backend pool resources	When the backend pool contains a resolvable FQDN, the DNS resolution is based on a private DNS zone or custom DNS server (if configured on the VNet), or it uses the default Azure-provided DNS.
Comply with all NSG restrictions for Application Gateway	NSGs are supported on Application Gateway subnet, but there are some restrictions. For instance, some communication with certain port ranges is prohibited. Make sure you understand the implications of those restrictions. For details, see Network security groups.

Recommendation	Benefit
Refrain from using UDRs on the Application gateway subnet	Using User Defined Routes (UDR) on the Application Gateway subnet cause some issues. Health status in the back-end might be unknown. Application Gateway logs and metrics might not get generated. We recommend that you don't use UDRs on the Application Gateway subnet so that you can view the back-end health, logs, and metrics. If your organizations require to use UDR in the Application Gateway subnet, please ensure you review the supported scenarios. For more information, see Supported user-defined routes.
Be aware of Application Gateway capacity changes when enabling WAF	When WAF is enabled, every request must be buffered by the Application Gateway until it fully arrives and check if the request matches with any rule violation in its core rule set and then forward the packet to the backend instances. For large file uploads (30MB+ in size), this can result in a significant latency. Because Application Gateway capacity requirements are different with WAF, we do not recommend enabling WAF on Application Gateway without proper testing and validation.

For more suggestions, see Principles of the security pillar.

Azure Advisor helps you ensure and improve continuity of your business-critical applications. Review the Azure Advisor recommendations.

Policy definitions

- Web Application Firewall (WAF) should be enabled for Application Gateway . Deploy Azure Web Application Firewall (WAF) in front of public facing web applications for additional inspection of incoming traffic. Web Application Firewall (WAF) provides centralized protection of your web applications from common exploits and vulnerabilities such as SQL injections, Cross-Site Scripting, local and remote file executions. You can also restrict access to your web applications by countries, IP address ranges, and other http(s) parameters via custom rules.
- Web Application Firewall (WAF) should use the specified mode for Application Gateway . Mandates the use of 'Detection' or 'Prevention' mode to be active on all Web Application Firewall policies for Application Gateway.
- Azure DDoS Protection should be enabled . DDoS protection should be enabled for all virtual networks with a subnet that is part of an application gateway with a public IP.

All built-in policy definitions related to Azure Networking are listed in Built-in policies - Network.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. We recommend you review the Cost optimization design principles.

Design checklist

- ✓ Familiarize yourself with Application Gateway pricing
- ✓ Review underutilized resources
- ✓ Stop Application Gateway instances that are not in use
- ✓ Have a scale-in and scale-out policy
- ✓ Review consumption metrics across different parameters

Recommendations

Explore the following table of recommendations to optimize your Application Gateway configuration for Cost optimization.

Recommendation	Benefit
Familiarize yourself with Application Gateway pricing	For information about Application Gateway pricing, see Understanding Pricing for Azure Application Gateway and Web Application Firewall. You can also leverage the Pricing calculator . Ensure that the options are adequately sized to meet the capacity demand and deliver expected performance without wasting resources.
Review underutilized resources	Identify and delete Application Gateway instances with empty backend pools to avoid unnecessary costs.
Stop Application Gateway instances when not in use	You aren't billed when Application Gateway is in the stopped state. Continuously running Application Gateway instances can incur extraneous costs. Evaluate usage patterns and stop instances when you don't need them. For example, usage after business hours in Dev/Test environments is expected to be low. See these articles for information about how to stop and start instances. - Stop-AzApplicationGateway - Start-AzApplicationGateway

Recommendation	Benefit
Have a scale-in and scale-out policy	A scale-out policy ensures that there will be enough instances to handle incoming traffic and spikes. Also, have a scale-in policy that makes sure the number of instances are reduced when demand drops. Consider the choice of instance size. The size can significantly impact the cost. Some considerations are described in the Estimate the Application Gateway instance count. For more information, see What is Azure Application Gateway v2?
Review consumption metrics across different parameters	You're billed based on metered instances of Application Gateway based on the metrics tracked by Azure. Evaluate the various metrics and capacity units and determine the cost drivers. For more information, see Azure Cost Management and Billing. The following metrics are key for Application Gateway. This information can be used to validate that the provisioned instance count matches the amount of incoming traffic. - Estimated Billed Capacity Units - Fixed Billable Capacity Units - Current Capacity Units For more information, see Application Gateway metrics. Make sure you account for bandwidth costs. For details, see Traffic across billing zones and regions.

For more suggestions, see Principles of the cost optimization pillar.

Azure Advisor helps you ensure and improve continuity of your business-critical applications. Review the Azure Advisor recommendations.

Operational excellence

Monitoring and diagnostics are crucial. Not only can you measure performance statistics but also use metrics troubleshoot and remediate issues quickly. We recommend you review the Operational excellence design principles.

Design checklist

- ✓ Monitor capacity metrics
- ✓ Enable diagnostics on Application Gateway and Web Application Firewall (WAF)
- ✓ Use Azure Monitor Network Insights

- ✓ Match timeout settings with the backend application
- ✓ Monitor Key Vault configuration issues using Azure Advisor
- ✓ Configure and monitor SNAT port limitations
- ✓ Consider SNAT port limitations in your design

Recommendations

Explore the following table of recommendations to optimize your Application Gateway configuration for Operational excellence.

Recommendation	Benefit
Monitor capacity metrics	Use these metrics as indicators of utilization of the provisioned Application Gateway capacity. We strongly recommend setting up alerts on capacity. For details, see Application Gateway high traffic support.
Troubleshoot using metrics	There are other metrics that can indicate issues either at Application Gateway or the backend. We recommend evaluating the following alerts: - Unhealthy Host Count - Response Status (dimension 4xx and 5xx) - Backend Response Status (dimension 4xx and 5xx) - Backend Last Byte Response Time - Application Gateway Total Time For more information, see Metrics for Application Gateway.
Enable diagnostics on Application Gateway and Web Application Firewall (WAF)	Diagnostic logs allow you to view firewall logs, performance logs, and access logs. Use these logs to manage and troubleshoot issues with Application Gateway instances. For more information, see Back-end health and diagnostic logs for Application Gateway.
Use Azure Monitor Network Insights	Azure Monitor Network Insights provides a comprehensive view of health and metrics for network resources, including Application Gateway. For additional details and supported capabilities for Application Gateway, see Azure Monitor Network insights.
Match timeout settings with the backend application	Ensure you have configured the IdleTimeout settings to match the listener and traffic characteristics of the backend application. The default value is set to four minutes and can be configured to a maximum of 30. For more information, see Load Balancer TCP Reset and Idle Timeout. For workload considerations, see Monitoring application health for reliability.

Recommendation	Benefit
Monitor Key Vault configuration issues using Azure Advisor	Application Gateway checks for the renewed certificate version in the linked Key Vault at every 4-hour interval. If it is inaccessible due to any incorrectly modified Key Vault configurations, it logs that error and pushes a corresponding Advisor recommendation. You must configure the Advisor alert to stay updated and fix such issues immediately to avoid any Control or Data plane related problems. To set an alert for this specific case, use the Recommendation Type as **Resolve Azure Key Vault issue for your Application Gateway**.
Consider SNAT port limitations in your design	SNAT port limitations are important for backend connections on the Application Gateway. There are separate factors that affect how Application Gateway reaches the SNAT port limit. For example, if the backend is a public IP address, it will require its own SNAT port. In order to avoid SNAT port limitations, you can increase the number of instances per Application Gateway, scale out the backends to have more IP addresses, or move your backends into the same virtual network and use private IP addresses for the backends. Requests per second (RPS) on the Application Gateway will be affected if the SNAT port limit is reached. For example, if an Application Gateway reaches the SNAT port limit, then it won't be able to open a new connection to the backend, and the request will fail.

For more suggestions, see Principles of the operational excellence pillar.

Azure Advisor helps you ensure and improve continuity of your business-critical applications. Review the Azure Advisor recommendations.

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. We recommend you review the Performance efficiency principles.

Design checklist

- ✓ Estimate the Application Gateway instance count
- ✓ Define the maximum instance count
- ✓ Define the minimum instance count
- ✓ Define Application Gateway subnet size
- ✓ Take advantage features for autoscaling and performance benefits

Recommendations

Explore the following table of recommendations to optimize your Application Gateway configuration for Performance efficiency.

Recommendation	Benefit
Estimate the Application Gateway instance count	Application Gateway v2 scales out based on many aspects, such as CPU, network throughput, current connections, and more. To determine the approximate instance count, factor in these metrics: Current compute units — Indicates CPU utilization. 1 Application Gateway instance is approximately 10 compute units. Throughput — Application Gateway instance can serve ~500 Mbps of throughput. This data depends on the type of payload. Consider this equation when calculating instance counts. $\text{Approximate instance count} = \mathit{max}\left(\frac{\text{Current compute units}}{10}, \frac{\text{Throughput in Mbps}}{500}\right)$ After you've estimated the instance count, compare that value to the maximum instance count. This will indicate how close you are to the maximum available capacity.
Define the minimum instance count	For Application Gateway v2 SKU, autoscaling takes some time (approximately six to seven minutes) before the additional set of instances is ready to serve traffic. During that time, if there are short spikes in traffic, expect transient latency or loss of traffic. We recommend that you set your minimum instance count to an optimal level. After you estimate the average instance count and determine your Application Gateway autoscaling trends, define the minimum instance count based on your application patterns. For information, see Application Gateway high traffic support. Check the Current Compute Units for the past one month. This metric represents the gateway's CPU utilization. To define the minimum instance count, divide the peak usage by 10. For example, if your average Current Compute Units in the past month is 50, set the minimum instance count to five.
Define the maximum instance count	We recommend 125 as the maximum autoscale instance count. Make sure the subnet that has the Application Gateway has sufficient available IP addresses to support the scale-up set of instances. Setting the maximum instance count to 125 has no cost implications because you're billed only for the consumed capacity.

Recommendation	Benefit
Define Application Gateway subnet size	Application Gateway needs a dedicated subnet within a virtual network. The subnet can have multiple instances of the deployed Application Gateway resource. You can also deploy other Application Gateway resources in that subnet, v1 or v2 SKU. Here are some considerations for defining the subnet size: - Application Gateway uses one private IP address per instance and another private IP address if a private front-end IP is configured. - Azure reserves five IP addresses in each subnet for internal use. - Application Gateway (Standard or WAF SKU) can support up to 32 instances. Taking 32 instance IP addresses + 1 private front-end IP + 5 Azure reserved, a minimum subnet size of /26 is recommended. Because the Standard_v2 or WAF_v2 SKU can support up to 125 instances, using the same calculation, a subnet size of /24 is recommended. - If you want to deploy additional Application Gateway resources in the same subnet, consider the additional IP addresses that will be required for their maximum instance count for both, Standard and Standard v2.
Take advantage features for autoscaling and performance benefits	The v2 SKU offers autoscaling to ensure that your Application Gateway can scale up as traffic increases. When compared to v1 SKU, v2 has capabilities that enhance the performance of the workload. For example, better TLS offload performance, quicker deployment and update times, zone redundancy, and more. For more information about autoscaling features, see Scaling Application Gateway v2 and WAF v2. If you are running v1 SKU gateways, consider migrating to the v2 SKU. For more information, see Migrate Azure Application Gateway and Web Application Firewall from v1 to v2.

Azure Advisor helps you ensure and improve continuity of your business-critical applications. Review the Azure Advisor recommendations.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost effectiveness, performance, and operational excellence of your Application Gateway.

Reliability

- Ensure application gateway fault tolerance
- Do not override hostname to ensure website integrity

Additional resources

Azure Architecture Center guidance

- Using API gateways in microservices
- Firewall and Application Gateway for virtual networks
- Protect APIs with Application Gateway and API Management
- IaaS: Web application with relational database
- Securely managed web applications
- Zero-trust network for web applications with Azure Firewall and Application Gateway

Next steps

- Deploy an Application Gateway to see how it works: Quickstart: Direct web traffic with Azure Application Gateway - Azure portal

Azure Well-Architected Framework review - Azure Firewall

Article • 01/20/2023

This article provides architectural best practices for Azure Firewall. The guidance is based on the five pillars of architecture excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

We assume that you have working knowledge of Azure Firewall and are well versed with its features. For more information, see Azure Firewall Standard features.

Prerequisites

- Understanding the Azure Well-Architected Framework pillars can help produce a high-quality, stable, and efficient cloud architecture. Review your workload by using the Well-Architected Framework review assessment.
- Use a reference architecture to review the considerations based on the guidance provided in this article. Start with Network-hardened web application with private connectivity to PaaS datastores and Implement a secure hybrid network.

Reliability

To learn how Azure Firewall supports a reliable workload, see the following articles:

- Introduction to Azure Firewall
- Quickstart: Deploy Azure Firewall with availability zones

Design checklist

As you make design choices for Azure Firewall, review the design principles for reliability.

✓ Deploy by using a secured virtual hub.
✓ Use a global Azure Firewall policy.
✓ Determine if you want to use third-party security as a service (SECaaS) providers.

Recommendations

Explore the following table of recommendations to optimize your Azure Firewall configuration for reliability.

Recommendation	Benefit
Use Azure Firewall Manager with Azure Virtual WAN to deploy and manage instances of Azure Firewall across Virtual WAN hubs or in hub virtual networks.	Easily create hub-and-spoke and transitive architectures with native security services for traffic governance and protection.
Create a global Azure Firewall policy to govern the security posture across global network environments. Assign the policy to all instances of Azure Firewall.	Allow for granular policies to meet the requirements of specific regions. Delegate incremental firewall policies to local security teams through role-based access control (RBAC).
Configure supported third-party software as a service (SaaS) security providers within Firewall Manager if you want to use these solutions to protect outbound connections.	You can use your familiar, best-in-breed, third-party SECaaS offerings to protect internet access for your users.
Deploy Azure Firewall across multiple availability zones for a higher service-level agreement (SLA).	Azure Firewall provides different SLAs when it's deployed in a single availability zone and when it's deployed in multizones. For more information, see SLA for Azure Firewall. For information about all Azure SLAs, see SLA summary for Azure services.
In multi-region environments, deploy an instance of Azure Firewall per region.	For workloads designed to be resistant to failures and fault tolerant, remember to consider that instances of Azure Firewall and Azure Virtual Network are regional resources.
Closely monitor Azure Firewall metrics to ensure this component of your solution is healthy.	Closely monitor metrics, especially SNAT port utilization, firewall health state, and throughput.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the Azure Advisor recommendations.

Security

Security is one of the most important aspects of any architecture. Azure Firewall is an intelligent firewall security service that provides threat protection for your cloud workloads running in Azure.

Design checklist

As you make design choices for Azure Firewall, review the design principles for security.

- ✓ Use a global Azure Firewall policy.
- ✓ Use threat intelligence.
- ✓ Use a DNS proxy.
- ✓ Direct network traffic through Azure Firewall.
- ✓ Validate spoke networks.
- ✓ Determine if you want to use third-party SECaaS providers.
- ✓ Use just-in-time (JIT) systems.
- ✓ Protect your hub virtual network with a DDoS protection plan.

Recommendations

Explore the following table of recommendations to optimize your Azure Firewall configuration for security.

Recommendation	Benefit
Create a global Azure Firewall policy to govern the security posture across global network environments. Assign the policy to all instances of Azure Firewall.	Allow for granular policies to meet the requirements of specific regions. Delegate incremental firewall policies to local security teams through RBAC.
Enable threat intelligence on Azure Firewall.	You can enable threat intelligence-based filtering for your firewall to alert and deny traffic from or to unknown IP addresses and domains. The IP addresses and domains are sourced from the Microsoft Threat Intelligence Feed. Intelligent Security Graph powers Microsoft threat intelligence and is used by multiple services, including Microsoft Defender for Cloud.
Enable Domain Name System (DNS) proxy and point the infrastructure DNS to Azure Firewall.	By default, Azure Firewall uses Azure DNS. Custom DNS allows you to configure Azure Firewall to use corporate DNS to resolve external and internal names.

Recommendation	Benefit
Configure the user-defined routes (UDR) to force traffic to Azure Firewall.	Configure UDRs to force traffic to Azure Firewall for `SpoketoSpoke`, `SpoketoInternet`, and `SpoketoHybrid` connectivity.
Validate if unnecessary peering exists between the hub virtual network where Azure Firewall is deployed, and other spoke virtual networks.	Helps to guarantee that undesired traffic isn't being sent to the Azure firewall or the hub network where the Azure Firewall is deployed.
Use security partner providers for third-party SECaaS offerings.	Security partner providers help filter internet traffic through a virtual private network or a branch to the internet.
Use JIT systems to control access to virtual machines (VMs) from the internet.	You can use Microsoft Defender for Cloud JIT to control access for clients that connect from the internet by using Azure Firewall.
Configure Azure Firewall in the forced tunneling mode to route all internet-bound traffic to a designated next hop instead of going directly to the internet.	Azure Firewall must have direct internet connectivity. If your **AzureFirewallSubnet** learns a default route to your on-premises network via the Border Gateway Protocol, you must configure Azure Firewall in the forced tunneling mode. Using the forced tunneling feature, you'll need another /26 address space for the Azure Firewall Management subnet. You're required to name it **AzureFirewallManagementSubnet**. If this is an existing Azure Firewall instance that can't be reconfigured in the forced tunneling mode, create a UDR with a 0.0.0.0/0 route. Set the **NextHopType** value as **Internet**. Associate it with **AzureFirewallSubnet** to maintain internet connectivity.
Set the public IP address to **None** to deploy a fully private data plane when you configure Azure Firewall in the forced tunneling mode.	When you deploy a new Azure Firewall instance, if you enable the forced tunneling mode, you can set the public IP address to **None** to deploy a fully private data plane. However, the management plane still requires a public IP for management purposes only. The internal traffic from virtual and on-premises networks won't use that public IP. For more about forced tunneling, see Azure Firewall forced tunneling.
Use fully qualified domain name (FQDN) filtering in network rules.	You can use FQDNs based on DNS resolution in Azure Firewall and firewall policies. This capability allows you to filter outbound traffic with any TCP/UDP protocol (including NTP, SSH, RDP, and more). You must enable the DNS Proxy option to use FQDNs in your network rules. To learn how it works, see Azure Firewall FQDN filtering in network rules.

Recommendation	Benefit
Use Azure Firewall Manager to create and associate a DDoS protection plan with your hub virtual network.	A DDoS protection plan provides enhanced mitigation features to defend your firewall from DDoS attacks. Azure Firewall Manager is an integrated tool to create your firewall infrastructure and DDoS protection plans. For more information, see Configure an Azure DDoS Protection Plan using Azure Firewall Manager.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the Azure Advisor recommendations.

Policy definitions

All internet traffic should be routed via your Azure Firewall. Protect your subnets from potential threats by restricting access to them with Azure Firewall or a supported next-generation firewall.

All built-in policy definitions related to Azure networking are listed in Built-in policies - Network.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies.

Design checklist

As you make design choices for Azure Firewall, review the design principles for cost optimization.

- ✓ Determine which firewall SKUs to deploy.
- ✓ Determine if some resources don't need 100% allocation.
- ✓ Determine where you can optimize firewall use across workloads.
- ✓ Monitor firewall usage to determine cost-effectiveness.
- ✓ Review Firewall Manager capabilities to determine potential operational efficiency.
- ✓ Determine the number of public IP addresses required.

Recommendations

Explore the following table of recommendations to optimize your Azure Firewall configuration for cost optimization.

Recommendation	Benefit
Deploy the Standard and Premium SKUs where appropriate.	The Standard option is usually enough for east-west traffic. Premium has the necessary extra features for north-south traffic, the forced tunneling feature, and many other features. For more information, see Azure Firewall Premium Preview features. Deploy mixed scenarios using the Standard and Premium options according to your needs.
Stop Azure Firewall deployments that don't need to run for 24 hours.	You might have development environments that are used only during business hours. For more information, see Deallocate and allocate Azure Firewall.
Share the same instance of Azure Firewall across multiple workloads and Azure Virtual Network.	You can use a central instance of Azure Firewall in the hub virtual network and share the same firewall across many spoke virtual networks that are connected to the same hub from the same region. Ensure there's no unexpected cross-region traffic as part of the hub-spoke topology.
Review underutilized Azure Firewall instances. Identify and delete unused Azure Firewall deployments.	To identify unused Azure Firewall deployments, start by analyzing the monitoring metrics and UDRs associated with subnets pointing to the firewall's private IP. Combine that information with other validations, such as if your instance of Azure Firewall has any rules (classic) for NAT, Network and Application, or even if the DNS Proxy setting is configured to **Disabled**, and with internal documentation about your environment and deployments. You can detect deployments that are cost-effective over time. For more information about monitoring logs and metrics, see Monitor Azure Firewall logs and metrics and SNAT port utilization.
Use Azure Firewall Manager and its policies to reduce operational costs, increase efficiency, and reduce management overhead.	Review your Firewall Manager policies, associations, and inheritance carefully. Policies are billed based on firewall associations. A policy with zero or one firewall association is free of charge. A policy with multiple firewall associations is billed at a fixed rate. For more information, see Pricing - Azure Firewall Manager.
Delete unused public IP addresses and use IP Groups to reduce your management overhead.	Validate whether all the associated public IP addresses are in use. If they aren't in use, disassociate and delete them. Use IP Groups to reduce your management overhead. Evaluate SNAT port utilization before removing any IP addresses. You'll only use the number of public IPs your firewall needs. For more information, see Monitor Azure Firewall logs and metrics and SNAT port utilization.

For more suggestions, see Principles of the Cost optimization pillar.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the Azure Advisor recommendations.

Operational excellence

Monitoring and diagnostics are crucial. You can measure performance statistics and metrics to troubleshoot and remediate issues quickly.

Design checklist

As you make design choices for Azure Firewall, review the design principles for operational excellence.

- ✓ Use logs for monitoring.
- ✓ Use tags when possible to allow traffic through the firewall.
- ✓ Use workbooks.
- ✓ Use the Azure Firewall connector in Microsoft Sentinel.

Recommendations

Explore the following table of recommendations to optimize your Azure Firewall configuration for operational excellence.

Recommendation	Benefit
Turn on logs for Azure Firewall.	You can monitor Azure Firewall by using firewall logs or workbooks. You can also use activity logs for auditing operations on Azure Firewall resources.
Use FQDN tags on Azure Firewall.	FQDN tags make it easy to allow known Azure service network traffic through your firewall. For example, say you want to allow Windows Update network traffic through your firewall. You create an application rule and use the Windows Update tag. Now network traffic from Windows Update can flow through your firewall.
Use workbooks in Azure Log Analytics.	Workbooks help visualize firewall logs.
Enable Azure Firewall connector in Microsoft Sentinel.	You can use Microsoft Sentinel to create detections and logic apps for Azure Firewall.

Recommendation	Benefit
Migrate Azure Firewall rules to Azure Firewall Manager policies for existing deployments.	For existing deployments, migrate Azure Firewall rules to Azure Firewall Manager policies. Use Azure Firewall Manager to centrally manage your firewalls and policies.
Monitoring capacity metrics are indicators of the utilization of provisioned Azure Firewall capacity.	Set alerts as needed to get notifications after reaching a threshold for any metric. For information about monitoring logs and metrics, see Monitor Azure Firewall logs and metrics.
Monitor other Azure Firewall logs and metrics for troubleshooting and set alerts.	Azure Firewall exposes a few other logs and metrics for troubleshooting that are suitable indicators of issues. Evaluate alerts based on the following list. • Application Rule log: Each new connection that matches one of your configured application rules results in a log for the accepted/denied connection. • Network Rule log: Each new connection that matches one of your configured network rules results in a log for the accepted/denied connection. • DNS Proxy log: This log tracks DNS messages to a DNS server configured using a DNS proxy.
Use diagnostics logs and policy analytics.	Diagnostic logs allow you to view Azure Firewall logs, performance logs, and access logs. You can use these logs in Azure to manage and troubleshoot your Azure Firewall instance. Policy analytics for Azure Firewall Manager allows you to start seeing rules and flows that match the rules and hit count for those rules. You can have full traffic visibility by watching what rule is in use and the traffic being matched.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the Azure Advisor recommendations.

Performance efficiency

Performance efficiency is the ability of your workload to scale to efficiently meet the demands placed on it by users.

Design checklist

As you make design choices for Azure Firewall, review the design principles for performance efficiency.

- ✓ Determine your SNAT port requirements and if you should deploy a NAT gateway.
- ✓ Plan load tests to test auto-scale performance in your environment.
- ✓ Plan network rule requirements and opportunities to summarize IP ranges.

Recommendations

Explore the following table of recommendations to optimize your Azure Firewall configuration for performance efficiency.

Recommendation	Benefit
If you'll need more than 512,000 SNAT ports, deploy a NAT gateway with Azure Firewall.	With a NAT gateway, you can scale up to more than 1 million ports. For more information, see Scale SNAT ports with Azure NAT gateway.
Create initial traffic that isn't part of your load tests 20 minutes before the test. Use diagnostics settings to capture scale-up and scale-down events. You can use the Azure Load Testing service to generate the initial traffic.	Allows the Azure Firewall instance to scale up its instances to the maximum.
Use IP Groups to summarize IP address ranges.	You can use IP Groups to summarize IP ranges, so you don't exceed 10,000 network rules. For each rule, Azure multiplies ports by IP addresses. So, if you have 1 rule with 4 IP address ranges and 5 ports, you'll consume 20 network rules.
Configure an Azure Firewall subnet (AzureFirewallSubnet) with a /26 address space.	Azure Firewall is a dedicated deployment in your virtual network. Within your virtual network, a dedicated subnet is required for the instance of Azure Firewall. Azure Firewall provisions more capacity as it scales. A /26 address space for its subnets ensures that the firewall has enough IP addresses available to accommodate the scaling. Azure Firewall doesn't need a subnet bigger than /26. The Azure Firewall subnet name must be **AzureFirewallSubnet**.

Azure Advisor helps you ensure and improve the continuity of your business-critical applications. Review the Azure Advisor recommendations.

Azure Advisor recommendations

Azure Advisor is a personalized cloud consultant that helps you follow best practices to optimize your Azure deployments. Here are some recommendations that can help you improve the reliability, security, cost-effectiveness, performance, and operational excellence of your instance of Azure Firewall.

- Create Azure Service Health alerts to be notified when Azure problems affect you
- Ensure you have access to Azure cloud experts when you need it
- Enable Traffic Analytics to view insights into traffic patterns across Azure resources
- Update your outbound connectivity protocol to Service Tags for Azure Site Recovery
- Follow just enough administration (least privilege principle)
- Protect your network resources with Microsoft Defender for Cloud

Additional resources

- Azure Firewall documentation
- Azure Firewall service limits, quotas, and constraints
- Azure security baseline for Azure Firewall

Azure Architecture Center guidance

- Azure Firewall architecture overview
- Use Azure Firewall to help protect an Azure Kubernetes Service (AKS) cluster
- Hub-spoke network topology in Azure
- Implement a secure hybrid network
- Network-hardened web application with private connectivity to PaaS datastores

Next step

Deploy an instance of Azure Firewall to see how it works:

- Tutorial: Deploy and configure Azure Firewall and policy by using the Azure portal

Azure Well-Architected Framework review - Azure ExpressRoute

Article • 11/18/2022

This article provides architectural best practice for Azure ExpressRoute. The guidance is based on the five pillars of the architecture excellence:

- Reliability
- Security
- Cost optimization
- Operational excellence
- Performance efficiency

We assume that you have working knowledge of Azure ExpressRoute and are well versed with all of its features. For more information, see Azure ExpressRoute.

Prerequisites

For context, consider reviewing a reference architecture that reflects these considerations in its design. We recommend that you start with Cloud Adoption Framework Ready methodology's guidance Connect to Azure and Architect for hybrid connectivity with Azure ExpressRoute. For low-code application architectures, we recommend reviewing Enabling ExpressRoute for Power Platform when planning and configuring ExpressRoute for use with Microsoft Power Platform.

Reliability

In the cloud, we acknowledge that failures happen. Instead of trying to prevent failures altogether, the goal is to minimize the effects of a single failing component. Use the following information to minimize down time to and from Azure when establishing connectivity using Azure ExpressRoute.

When discussing about reliability with Azure ExpressRoute it's important to taking into consideration bandwidth usage, physical layout of the network, and disaster recovery if there's failures. Azure ExpressRoute is capable of achieving these design considerations and have recommendations for each item in the checklist.

In the **design checklist** and **list of recommendations** below, information is presented in order for you to design a highly available network between your Azure environment and on-premises network.

Design checklist

As you make design choices for Azure ExpressRoute, review the design principles for adding reliability to the architecture.

- ✓ Select between ExpressRoute circuit or ExpressRoute Direct for business requirements.
- ✓ Configure a diverse physical layer network to the service provider.
- ✓ Configure ExpressRoute circuits with different service provider to have diverse routing paths.
- ✓ Configure Active-Active ExpressRoute connections between on-premises and Azure.
- ✓ Set up availability zone aware ExpressRoute Virtual Network Gateways.
- ✓ Configure ExpressRoute circuits in a different location than the on-premises network.
- ✓ Configure ExpressRoute Virtual Network Gateways in different regions.
- ✓ Configure site-to-site VPN as a backup to ExpressRoute private peering.
- ✓ Set up monitoring for ExpressRoute circuit and ExpressRoute Virtual Network Gateway health.
- ✓ Configure service health to receive ExpressRoute circuit maintenance notification.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for Reliability.

Recommendation	Benefit
Plan for ExpressRoute circuit or ExpressRoute Direct	During the initial planning phase, you want to decide whether you want to configure an ExpressRoute circuit or an ExpressRoute Direct connection. An ExpressRoute circuit allows a private dedicated connection into Azure with the help of a connectivity provider. ExpressRoute Direct allows you to extend on-premises network directly into the Microsoft network at a peering location. You also need to identify the bandwidth requirement and the SKU type requirement for your business needs.
Physical layer diversity	For better resiliency, plan to have multiple paths between the on-premises edge and the peering locations (provider/Microsoft edge locations). This configuration can be achieved by going through different service provider or through a different location from the on-premises network.
Plan for geo-redundant circuits	To plan for disaster recovery, set up ExpressRoute circuits in more than one peering locations. You can create circuits in peering locations in the same metro or different metro and choose to work with different service providers for diverse paths through each circuit. For more information, see Designing for disaster recovery and Designing for high availability.

Recommendation	Benefit
Plan for Active-Active connectivity	ExpressRoute dedicated circuits guarantee `99.95%` availability when an active-active connectivity is configured between on-premises and Azure. This mode provides higher availability of your Expressroute connection. It's also recommended to configure BFD for faster failover if there's a link failure on a connection.
Planning for Virtual Network Gateways	Create availability zone aware Virtual Network Gateway for higher resiliency and plan for Virtual Network Gateways in different region for disaster recovery and high availability.
Monitor circuits and gateway health	Set up monitoring and alerts for ExpressRoute circuits and Virtual Network Gateway health based on various metrics available.
Enable service health	ExpressRoute uses service health to notify about planned and unplanned maintenance. Configuring service health will notify you about changes made to your ExpressRoute circuits.

For more suggestions, see Principles of the reliability pillar.

Azure Advisor provides many recommendations for ExpressRoute circuits as they relate to reliability. For example, Azure Advisor can detect:

- ExpressRoute gateways in which only a single ExpressRoute circuit is deployed, instead of multiple. Multiple ExpressRoute circuits are recommended for add resiliency for the peering location.
- ExpressRoute circuits that aren't being observed by Connection Monitor, as end-to-end monitoring of your ExpressRoute circuit is critical for reliability insights.
- Network topologies involving multiple peering locations that would benefit from ExpressRoute Global Reach to improve disaster recovery designs for on-premises connectivity to account for unplanned connectivity loss.

Security

Security is one of the most important aspects of any architecture. ExpressRoute provides features to employ both the principle of least privilege and defense-in-defense. We recommend you review the Security design principles.

Design checklist

✓ Configure Activity log to send logs to archive.

- ✓ Maintain an inventory of administrative accounts with access to ExpressRoute resources.
- ✓ Configure MD5 hash on ExpressRoute circuit.
- ✓ Configure MACSec for ExpressRoute Direct resources.
- ✓ Encrypt traffic over private peering and Microsoft peering for virtual network traffic.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for security.

Recommendation	Benefit
Configure Activity log to send logs to archive	Activity logs provide insights into operations that were performed at the subscription level for ExpressRoute resources. With Activity logs, you can determine who and when an operation was performed at the control plane. Data retention is only 90 days and required to be stored in Log Analytics, Event Hubs or a storage account for archive.
Maintain inventory of administrative accounts	Use Azure RBAC to configure roles to limit user accounts that can add, update, or delete peering configuration on an ExpressRoute circuit.
Configure MD5 hash on ExpressRoute circuit	During configuration of private peering or Microsoft peering, apply an MD5 hash to secure messages between the on-premises route and the MSEE routers.
Configure MACSec for ExpressRoute Direct resources	Media Access Control security is a point-to-point security at the data link layer. ExpressRoute Direct supports configuring MACSec to prevent security threats to protocols such as ARP, DHCP, LACP not normally secured on the Ethernet link. For more information on how to configure MACSec, see MACSec for ExpressRoute Direct ports.
Encrypt traffic using IPsec	Configure a Site-to-site VPN tunnel over your ExpressRoute circuit to encrypt data transferring between your on-premises network and Azure virtual network. You can configure a tunnel using private peering or using Microsoft peering.

For more suggestions, see Principles of the security pillar.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. We recommend you review the Cost optimization design principle and Plan and manage costs for Azure ExpressRoute.

Design checklist

- ✓ Familiarize yourself with ExpressRoute pricing.
- ✓ Determine the ExpressRoute circuit SKU and bandwidth required.
- ✓ Determine the ExpressRoute virtual network gateway size required.
- ✓ Monitor cost and create budget alerts.
- ✓ Deprovision ExpressRoute circuits no longer in use.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for Cost optimization.

Recommendation	Benefit
Familiarize yourself with ExpressRoute pricing	For information about ExpressRoute pricing, see Understand pricing for Azure ExpressRoute. You can also use the Pricing calculator. Ensure that the options are adequately sized to meet the capacity demand and deliver expected performance without wasting resources.
Determine SKU and bandwidth required	The way you're charged for your ExpressRoute usage varies between the three different SKU types. With Local SKU, you're automatically charged with an Unlimited data plan. With Standard and Premium SKU, you can select between a Metered or an Unlimited data plan. All ingress data are free of charge except when using the Global Reach add-on. It's important to understand which SKU types and data plan works best for your workload to best optimize cost and budget. For more information resizing ExpressRoute circuit, see upgrading ExpressRoute circuit bandwidth.
Determine the ExpressRoute virtual network gateway size	ExpressRoute virtual network gateways are used to pass traffic into a virtual network over private peering. Review the performance and scale needs of your preferred Virtual Network Gateway SKU. Select the appropriate gateway SKU on your on-premises to Azure workload.
Monitor cost and create budget alerts	Monitor the cost of your ExpressRoute circuit and create alerts for spending anomalies and overspending risks. For more information, see Monitoring ExpressRoute costs.
Deprovision and delete ExpressRoute circuits no longer in use.	ExpressRoute circuits are charged from the moment they're created. To reduce unnecessary cost, deprovision the circuit with the service provider and delete the ExpressRoute circuit from your subscription. For steps on how to remove an ExpressRoute circuit, see Deprovisioning an ExpressRoute circuit.

For more suggestions, see Principles of the cost optimization pillar.

Azure Advisor can detect ExpressRoute circuits that have been deployed for a significant time but have a provider status of *Not Provisioned*. Circuits in this state aren't operational; and removing the unused resource will reduce unnecessary costs.

Operational excellence

Monitoring and diagnostics are crucial. Not only can you measure performance statistics but also use metrics troubleshoot and remediate issues quickly. We recommend you review the Operational excellence design principles.

Design checklist

- ✓ Configure connection monitoring between your on-premises and Azure network.
- ✓ Configure Service Health for receiving notification.
- ✓ Review metrics and dashboards available through ExpressRoute Insights using Network Insights.
- ✓ Review ExpressRoute resource metrics.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for Operational excellence.

Recommendation	Benefit
Configure connection monitoring	Connection monitoring allows you to monitor connectivity between your on-premises resources and Azure over the ExpressRoute private peering and Microsoft peering connection. Connection monitor can detect networking issues by identifying where along the network path the problem is and help you quickly resolve configuration or hardware failures.
Configure Service Health	Set up Service Health notifications to alert when planned and upcoming maintenance is happening to all ExpressRoute circuits in your subscription. Service Health also displays past maintenance along with RCA if an unplanned maintenance were to occur.

Recommendation	Benefit
Review metrics with Network Insights	ExpressRoute Insights with Network Insights allow you to review and analyze ExpressRoute circuits, gateways, connections metrics and health dashboards. ExpressRoute Insights also provide a topology view of your ExpressRoute connections where you can view details of your peering components all in a single place. Metrics available: - Availability - Throughput - Gateway metrics
Review ExpressRoute resource metrics	ExpressRoute uses Azure Monitor to collect metrics and create alerts base on your configuration. Metrics are collected for ExpressRoute circuits, ExpressRoute gateways, ExpressRoute gateway connections, and ExpressRoute Direct. These metrics are useful for diagnosing connectivity problems and understanding the performance of your ExpressRoute connection.

For more suggestions, see Principles of the operational excellence pillar.

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands placed on it by users in an efficient manner. We recommend you review the Performance efficiency principles.

Design checklist

- ✓ Test ExpressRoute gateway performance to meet work load requirements.
- ✓ Increase the size of the ExpressRoute gateway.
- ✓ Upgrade the ExpressRoute circuit bandwidth.
- ✓ Enable ExpressRoute FastPath for higher throughput.
- ✓ Monitor the ExpressRoute circuit and gateway metrics.

Recommendations

Explore the following table of recommendations to optimize your ExpressRoute configuration for performance efficiency.

Recommendation	Benefit

Recommendation	Benefit
Test ExpressRoute gateway performance to meet work load requirements.	Use Azure Connectivity Toolkit to test performance across your ExpressRoute circuit to understand bandwidth capacity and latency of your network connection.
Increase the size of the ExpressRoute gateway.	Upgrade to a higher gateway SKU for improved throughput performance between on-premises and Azure environment.
Upgrade ExpressRoute circuit bandwidth	Upgrade your circuit bandwidth to meet your work load requirements. Circuit bandwidth is shared between all virtual networks connected to the ExpressRoute circuit. Depending on your work load, one or more virtual networks can use up all the bandwidth on the circuit.
Enable ExpressRoute FastPath for higher throughput	If you're using an Ultra performance or an ErGW3AZ virtual network gateway, you can enable FastPath to improve the data path performance between your on-premises network and Azure virtual network.
Monitor ExpressRoute circuit and gateway metrics	Set up alerts base on ExpressRoute metrics to proactively notify you when a certain threshold is met. These metrics are useful to understand anomalies that can happen with your ExpressRoute connection such as outages and maintenance happening to your ExpressRoute circuits.

For more suggestions, see Principles of the performance efficiency pillar.

Azure Advisor will offer a recommendation to upgrade your ExpressRoute circuit bandwidth to accommodate usage when your circuit has recently been consuming over 90% of your procured bandwidth. If your traffic exceeds your allocated bandwidth, you'll experience dropped packets, which can lead to significant performance or reliability impact.

Azure Policy

Azure Policy doesn't provide any built-in policies for ExpressRoute, but custom policies can be created to help govern how ExpressRoute circuits should match your desired end state, such as SKU choice, peering type, peering configurations and so on.

Additional resources

Cloud Adoption Framework guidance

- Traditional Azure network topology
- Virtual WAN network topology (Microsoft-managed)

Next steps

Configure an ExpressRoute circuit or ExpressRoute Direct port to establish communication between your on-premises network and Azure.

API Management and reliability

Article • 11/30/2022

Learn how to use API Management to publish APIs to external, partner, and employee developers securely and at scale. This networking service is a hybrid, multicloud management platform for APIs across all environments.

Components include:

- API gateway
- Management plane
- Developer portal

For more information, reference About API Management.

To understand how API Management can increase reliability for your workload, reference the following topics:

- Availability zone support for Azure API Management
- How to deploy an Azure API Management service instance to multiple Azure regions
- How to implement disaster recovery using service backup and restore in Azure API Management

Checklist

Have you configured API Management with reliability in mind?

- ✓ Secure the communication between API Management and your backend.
- ✓ Ensure that each party has its own credential when exposing APIs to third parties.
- ✓ Ensure you set quotas and rate limits when exposing APIs to third parties.
- ✓ Evaluate the need for response caching.
- ✓ Plan a backup and restore process for your API Management instance.
- ✓ Configure multiple Azure regions in your API Management service.
- ✓ Implement a strategy to ensure availability during an outage or disaster affecting an Azure region.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your API Management service:

Recommendation	Description
Ensure you set quotas and rate limits when exposing APIs to third parties.	Protect backend services and reduce the load placed on an API Management scale unit. Rate limiting policies can be applied at Global, Product, API, and Operation levels to provide rate limit customization applied to API consumers.
Evaluate the need for response caching.	Response caching can reduce API latency and bandwidth consumption. Response caching reduces the load placed on the backend APIs leading to improved performance, user experience, and reduced solution cost.
Plan a backup and restore process for your API Management instance.	Consider taking regular backups of your API Management service so that you can easily restore it in another region. Your recovery time objective may require that a standby is deployed in a secondary region. It is a good practice to take regular backups to recreate the service due to unforeseen loss or misconfiguration of the service. Regular backups allow you to replicate changes between your primary and standby instances.
Configure multiple Azure regions in your API Management service.	Configure your API Management service with multiple regions to provide high-availability support in case an Azure region experiences downtime or a disaster scenario. Configuring multiple regions also reduces API call latency because calls can be routed to the nearest region.
Implement a strategy to ensure availability during an outage or disaster affecting an Azure region.	Consider using Azure Traffic Manager, Azure Front Door, or Azure DNS to enable access to multiple regional deployments of API Management. Using these services ensures you can still serve requests due to an outage or disaster. Requirements include syncing configurations between these individual Standard instances.

Next step

API Management and cost optimization

API Management and cost optimization

Article • 11/30/2022

Learn how to use API Management to publish APIs to external, partner, and employee developers securely and at scale. This networking service is a hybrid, multicloud management platform for APIs across all environments.

Components include:

- API gateway
- Management plane
- Developer portal

For more information, reference About API Management.

To understand how API Management supports cost optimization for your workload, reference the following topics:

- Automatically scale an Azure API Management instance
- Use a virtual network with Azure API Management

Checklist

Have you configured API Management with cost optimization in mind?

- ✓ Configure autoscaling where appropriate.
- ✓ Consider which features you need all the time.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your API Management service:

Recommendation	Description
Configure autoscaling where appropriate.	Consider scaling your API Management instance up or down to control costs. You can configure API Management with Autoscale based on a metric or a specific count. Costs depend upon the number of units, which determines throughput in requests per seconds (RPS). An autoscaled API Management instance switches between scale units appropriate for RPS numbers during a specific time window. Autoscaling helps to achieve balance between cost optimization and performance.

Recommendation	Description
Consider which features you need all the time.	Consider switching between Basic, Standard, and Premium tiers. If a workload does not need features available in higher tiers, then consider switching to a lower tier. As an example, a workload may need just `1GB` of cache during off-peak periods compared to `5GB` of cache during peak periods. Costs associated with such a workload can be reduced by switching from a Premium to Standard tier during off-peak periods and back to a Premium tier during peak periods. This process can be automated as a job using Set-AzApiManagement cmdlet. Refer to API Management pricing about features available in different API Management tiers.

Next step

API Management and operational excellence

API Management and operational excellence

Article • 11/30/2022

Learn how to use API Management to publish APIs to external, partner, and employee developers securely and at scale. This networking service is a hybrid, multicloud management platform for APIs across all environments.

Components include:

- API gateway
- Management plane
- Developer portal

For more information, reference About API Management.

To understand how API Management supports operational excellence, reference the following topics:

- Managing Azure API Management using Azure Automation
- Observability in Azure API Management

Checklist

Have you configured API Management with operational excellence in mind?

✓ Secure the communication between API Management and your backend.
✓ Ensure that each party has its own credential when exposing APIs to third parties.
✓ Ensure you set quotas and rate limits when exposing APIs to third parties.
✓ Understand the Microsoft REST API design and architecture guidance.
✓ Enable versioning of APIs to maintain backwards compatibility while adding other features.
✓ Use the API Management Versioning and Revisions features to implement API versioning.
✓ Understand the API import restrictions in API Management.
✓ Understand the Event logging feature.
✓ Trace calls in Azure API Management to help with debugging and testing.
✓ Configure logging using Azure Monitor for the API Management service.
✓ Choose the right modes to access private site connections.
✓ Evaluate firewall rules and IP allowlists based on the API Management public IP address.

Configuration recommendations

Consider the following recommendations for operational excellence when configuring your API Management service:

Recommendation	Description
Ensure you set quotas and rate limits when exposing APIs to third parties.	Protect backend services and reduce the load placed on an API Management scale unit. Rate limiting policies can be applied at Global, Product, API, and Operation levels to provide rate limit customization applied to API consumers.
Understand the Microsoft REST API design and architecture guidance.	Follow standards and best practices when using the REST API. Following best practices enables maximum compatibility across platforms and implementations. Review the REST API Guidelines and API Design guidance.
Understand the API import restrictions in API Management.	Every effort is made to ensure the API import process runs smoothly, which includes requiring no customizations. Some scenarios impose restrictions that will require modification to the import source. Applies to both REST and SOAP services. Reference Policy Restrictions for the current API Import restrictions.
Understand the Event logging feature.	Supports event logging to an Azure event hub to perform near real-time analysis. This feature integrates with external logging, security information and event management (SIEM) solutions, or analyzing API usage in near real time.
Trace calls in Azure API Management to help with debugging and testing.	Tracing must be enabled on the subscription used to make the request. Tracing is enabled on a request-by-request basis using the Ocp-Apim-Trace header value. API Tracing is also built into the admin portal and is enabled by default when testing APIs from the portal.
Configure logging using Azure Monitor for the API Management service.	Logs can be sent to a Logs Analytics workspace to enable complex querying and analysis. Metrics can be ingested for longer term analysis. All data is then surfaced using Azure Monitor. It is possible to integrate Application Insights for Application Performance Management.
Choose the right modes to access private site connections.	Supports Virtual Network integration in internal and external mode.

Recommendation	Description
Evaluate firewall rules and IP allowlists based on the API Management public IP address.	A fixed public IP address is available for the lifetime of the service with the Basic, Developer, Standard, and Premium plans for API Management.

Next step

Reliability and Azure Firewall

Reliability and Azure Front Door

Article • 11/30/2022

A scalable and secure entry point, Azure Front Door provides fast delivery of your global web applications. Front Door uses the Microsoft global edge network to create fast, secure, and widely scalable web applications.

Key features include:

- Accelerated application performance by using split TCP-based anycast protocol.
- Intelligent health probe monitoring for backend resources.
- URL-path based routing for requests.
- Enables hosting of multiple websites for efficient application infrastructure.
- Cookie-based session affinity.

For more key features and information, reference Why use Azure Front Door?

To understand how Azure Front Door creates a more reliable workload, reference the following topics:

- Selecting the Front Door environment for traffic routing (Anycast)
- Front Door routing methods

Checklist

Have you configured Azure Front Door with reliability in mind?

- ✓ Use WAF policies in Front Door. Lock down Application Gateway to receive traffic only from Azure Front Door when using Azure Front Door and Application Gateway to protect `HTTP/S` applications.
- ✓ Use Azure Front Door Web Application Firewall (WAF) policies to provide global protection across Azure regions for inbound `HTTP/S` connections to a *Landing Zone*.
- ✓ Create a rule to block access to the health endpoint from the internet.
- ✓ Ensure that the connection to the back-end is re-encrypted.
- ✓ Evaluate the four traffic routing configurations in Azure Front Door.

Configuration recommendations

Consider the following recommendation to optimize reliability when configuring Azure Front Door:

Recommendation	Description
Use WAF policies in Front Door. Lock down Application Gateway to receive traffic only from Azure Front Door when using Azure Front Door and Application Gateway to protect `HTTP/S` applications.	Certain scenarios can force a customer to implement rules specifically on AppGateway: For example, if ModSec Core Rule Set (CRS) `2.2.9`, CRS `3.0`, or CRS `3.1` rules are required, rules can be only implemented on AppGatway. Rate-limiting and geo-filtering are available only on Azure Front Door, not on AppGateway. Instructions on how to lock down traffic can be found at Frequently asked questions for Azure Front Door
Ensure that the connection to the back-end is re-encrypted.	Front Door doesn't support SSL passthrough. Front Door must hold the certificate to terminate the encrypted inbound connection.
Evaluate the four traffic routing configurations in Azure Front Door.	The Front Door service supports various traffic-routing methods to determine how to route your `HTTP/HTTPS` traffic to the various service endpoints.

Next step

Security and Azure Front Door

Security and Azure Front Door

Article • 11/30/2022

A scalable and secure entry point, Azure Front Door provides fast delivery of your global web applications. Front Door uses the Microsoft global edge network to create fast, secure, and widely scalable web applications.

Key features include:

- Accelerated application performance by using split TCP-based anycast protocol.
- Intelligent health probe monitoring for backend resources.
- URL-path based routing for requests.
- Enables hosting of multiple websites for efficient application infrastructure.
- Cookie-based session affinity.

For more key features and information, reference Why use Azure Front Door?

To understand how Azure Front Door creates a more secure workload, reference the following topics:

- Security baseline for Azure Front Door
- DDoS protection on Front Door
- End-to-end Transport Layer Security (TLS) with Azure Front Door

Checklist

Have you configured Azure Front Door with security in mind?

✓ Consider using geo-filtering in Azure Front Door.

Configuration recommendations

Consider the following recommendation to optimize security when configuring Azure Front Door:

Recommendation	Description
Consider using geo-filtering in Azure Front Door.	The Front Door service responds to user requests regardless of the location of the user making the request.

Next step

Operational excellence and Azure Front Door

Operational excellence and Azure Front Door

Article • 11/30/2022

A scalable and secure entry point, Azure Front Door provides fast delivery of your global web applications. Front Door uses the Microsoft global edge network to create fast, secure, and widely scalable web applications.

Key features include:

- Accelerated application performance by using split TCP-based anycast protocol.
- Intelligent health probe monitoring for backend resources.
- URL-path based routing for requests.
- Enables hosting of multiple websites for efficient application infrastructure.
- Cookie-based session affinity.

For more key features and information, reference Why use Azure Front Door?

To understand how Azure Front Door supports operational excellence, reference the following topics:

- How Front Door determines backend health
- Monitoring metrics and logs in Azure Front Door
- Front Door Standard/Premium (Preview) Logging

Checklist

Have you configured Azure Front Door with operational excellence in mind?

- ✓ Use Web Application Firewall (WAF) policies in Front Door. Lock down Application Gateway to receive traffic only from Azure Front Door when using Azure Front Door and Application Gateway to protect `HTTP/S` applications.
- ✓ Use Azure Front Door Web Application Firewall (WAF) policies to provide global protection across Azure regions for inbound `HTTP/S` connections to a *Landing Zone*.
- ✓ Create a rule to block access to the health endpoint from the internet.
- ✓ Ensure that the connection to the back-end is re-encrypted.

Configuration recommendations

Consider the following recommendation to optimize operational excellence when configuring Azure Front Door:

Recommendation	Description
Use Web Application Firewall (WAF) policies in Front Door. Lock down Application Gateway to receive traffic only from Azure Front Door when using Azure Front Door and Application Gateway to protect `HTTP/S` applications.	Certain scenarios can force a customer to implement rules specifically on AppGateway: For example, if ModSec Core Rule Set (CRS) `2.2.9`, CRS `3.0`, or CRS `3.1` rules are required, rules can be only implemented on AppGatway. Rate-limiting and geo-filtering are available only on Azure Front Door, not on AppGateway. Instructions on how to lock down traffic can be found at Frequently asked questions for Azure Front Door
Ensure that the connection to the back-end is re-encrypted.	Front Door doesn't support SSL passthrough. Front Door must hold the certificate to terminate the encrypted inbound connection.

Next step

Reliability and Network Virtual Appliances

Reliability and Network Virtual Appliances (NVA)

Article • 11/30/2022

Network Virtual Appliances (NVA) are typically used to control the flow of traffic between network segments classified with different security levels, for example between a perimeter network (also known as DMZ, demilitarized zone, and screened subnet) and the public internet.

Examples of NVAs include:

- Network firewalls
- Layer-4 reverse-proxies
- Internet Protocol Security (IPsec) Virtual Private Network (VPN) endpoints
- Web-based reverse-proxies
- Internet proxies
- Layer-7 load balancers

For more information about Network Virtual Appliances, reference Deploy highly available NVAs.

To understand how NVAs support a reliable workload, reference the following topics:

- Scenario: Route traffic through an NVA
- Scenario: Route traffic through NVAs by using custom settings
- Use L7 load balancers

Checklist

Have you configured your Network Virtual Appliances (NVA) with reliability in mind?

- ✓ NVAs should be deployed within a *Landing Zone* or *solution-level* Virtual Network.
- ✓ For Virtual Wide Area Network (VWAN) topologies, deploy the NVAs to a separate Virtual Network (such as, NVA VNet). Connect the NVA to the regional Virtual WAN Hub and to the *Landing Zones* that require access to NVAs.
- ✓ For non-Virtual Wide Are Network (WAN) topologies, deploy the third-party NVAs in the central Hub Virtual Network (VNet).

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your Network Virtual Appliances (NVA):

Recommendation	Description
NVAs should be deployed within a *Landing Zone* or *solution-level* Virtual Network.	If third-party NVAs are required for inbound `HTTP/S` connections, deploy NVAs together with the applications that they're protecting and exposing to the internet.
For Virtual Wide Area Network (VWAN) topologies, deploy the NVAs to a separate Virtual Network (such as, NVA VNet). Connect the NVA to the regional Virtual WAN Hub and to the *Landing Zones* that require access to NVAs.	If third-party NVAs are required for east-west or south-north traffic protection and filtering, reference Scenario: Route traffic through an NVA.
For non-Virtual Wide Area Network (WAN) topologies, deploy the third-party NVAs in the central Hub Virtual Network (VNet).	If third-party NVAs are required for east-west or south-north traffic protection and filtering, deploy the third-party NVAs in the central Hub Virtual Network.

Next step

Cost optimization and Network Virtual Appliances (NVA)

Cost optimization and Network Virtual Appliances (NVA)

Article • 11/30/2022

Network Virtual Appliances (NVA) are typically used to control the flow of traffic between network segments classified with different security levels, for example between a perimeter network (also known as DMZ, demilitarized zone, and screened subnet) and the public internet.

Examples of NVAs include:

- Network firewalls
- Layer-4 reverse-proxies
- Internet Protocol Security (IPsec) Virtual Private Network (VPN) endpoints
- Web-based reverse-proxies
- Internet proxies
- Layer-7 load balancers

For more information about Network Virtual Appliances, reference Deploy highly available NVAs.

Design considerations

When deploying a Network Virtual Appliance (NVA), keep in mind the following design considerations:

- There's a difference between using a third-party app (NVA) and using an Azure native service (Firewall or Application Gateway).
- With managed Platform as a Service (PaaS) services such as Azure Firewall or Application Gateway, Microsoft handles the management of the service and the underlying infrastructure. Using NVAs, which usually have to be deployed on Virtual Machines or Infrastructure as a Service (IaaS), the customer has to handle the management operations (such as patching and updating) of that Virtual Machine and the appliance on top. Managing third-party services also involves using specific vendor tools making integration difficult.

Next step

Operational excellence and Network Virtual Appliances (NVA)

Operational excellence and Network Virtual Appliances (NVA)

Article • 11/30/2022

Network Virtual Appliances (NVA) are typically used to control the flow of traffic between network segments classified with different security levels, for example between a perimeter network (also known as DMZ, demilitarized zone, and screened subnet) and the public internet.

Examples of NVAs include:

- Network firewalls
- Layer-4 reverse-proxies
- Internet Protocol Security (IPsec) Virtual Private Network (VPN) endpoints
- Web-based reverse-proxies
- Internet proxies
- Layer-7 load balancers

For more information about Network Virtual Appliances, reference Deploy highly available NVAs.

To understand how NVAs promote operational excellence, reference the following topics:

- Scenario: Route traffic through an NVA
- Scenario: Route traffic through NVAs by using custom settings
- Gateway Load Balancer

Checklist

Have you configured your Network Virtual Appliances (NVA) with operational excellence in mind?

- ✓ NVAs should be deployed within a *Landing Zone* or *solution-level* Virtual Network.
- ✓ For Virtual Wide Area Network (VWAN) topologies, deploy the NVAs to a separate Virtual Network (such as, NVA VNet). Connect the NVA to the regional Virtual WAN Hub and to the *Landing Zones* that require access to NVAs.
- ✓ For non-Virtual Wide Are Network (WAN) topologies, deploy the third-party NVAs in the central Hub Virtual Network (VNet).

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring your Network Virtual Appliances (NVA):

Recommendation	Description
NVAs should be deployed within a *Landing Zone* or *solution-level* Virtual Network.	If third-party NVAs are required for inbound `HTTP/S` connections, deploy NVAs together with the applications that they're protecting and exposing to the internet.
For Virtual Wide Area Network (VWAN) topologies, deploy the NVAs to a separate Virtual Network (such as, NVA VNet). Connect the NVA to the regional Virtual WAN Hub and to the *Landing Zones* that require access to NVAs.	If third-party NVAs are required for east-west or south-north traffic protection and filtering, reference Scenario: Route traffic through an NVA.
For non-Virtual Wide Area Network (WAN) topologies, deploy the third-party NVAs in the central Hub Virtual Network (VNet).	If third-party NVAs are required for east-west or south-north traffic protection and filtering, deploy the third-party NVAs in the central Hub Virtual Network.

Next step

Reliability and Network connectivity

Reliability and Network connectivity

Article • 11/30/2022

Network connectivity includes three Azure models for private network connectivity:

- VNet injection
- VNet service endpoints
- Private Link

VNet injection applies to services that are deployed specifically for you, such as:

- Azure Kubernetes Service (AKS) nodes
- SQL Managed Instance
- Virtual Machines

These resources connect directly to your virtual network.

Virtual Network (VNet) service endpoints provide secure and direct connectivity to Azure services. These service endpoints use an optimized route over the Azure network. Service endpoints enable private IP addresses in the VNet to reach the endpoint of an Azure service without needing a public IP address on the VNet.

Private Link provides dedicated access using private IP addresses to Azure PaaS instances, or custom services behind an Azure Load Balancer Standard.

Design considerations

Network connectivity includes the following design considerations related to a reliable workload:

- Use Private Link, where available, for shared Azure PaaS services. Private Link is generally available for several services and is in public preview for numerous ones.

- Access Azure PaaS services from on-premises through ExpressRoute private peering.

- Use either virtual network injection for dedicated Azure services or Azure Private Link for available shared Azure services. To access Azure PaaS services from on-premises when virtual network injection or Private Link isn't available, use ExpressRoute with Microsoft peering. This method avoids transiting over the public internet.

- Use virtual network service endpoints to secure access to Azure PaaS services from within your virtual network. Use virtual network service endpoints only when Private Link isn't available and there are no concerns with unauthorized movement of data.

- Service Endpoints don't allow a PaaS service to be accessed from on-premises networks. Private Endpoints do.

- To address concerns about unauthorized movement of data with service endpoints, use network-virtual appliance (NVA) filtering. You can also use virtual network service endpoint policies for Azure Storage.

- The following native network security services are fully managed services. Customers don't incur the operational and management costs associated with infrastructure deployments, which can become complex at scale:
  - Azure Firewall
  - Application Gateway
  - Azure Front Door

- PaaS services are typically accessed over public endpoints. The Azure platform provides capabilities to secure these endpoints or make them entirely private.

- You can also use third-party network-virtual appliances (NVAs) if the customer prefers them for situations where native services don't satisfy specific requirements.

Checklist

Have you configured Network connectivity with reliability in mind?

✓ Don't implement forced tunneling to enable communication from Azure to Azure resources.
✓ Unless you use network virtual appliance (NVA) filtering, don't use virtual network service endpoints when there are concerns about unauthorized movement of data.
✓ Don't enable virtual network service endpoints by default on all subnets.

Next step

Cost optimization and Network connectivity

Cost optimization and Network connectivity

Article • 11/30/2022

Network connectivity includes three Azure models for private network connectivity:

- VNet injection
- VNet service endpoints
- Private Link

VNet injection applies to services that are deployed specifically for you, such as:

- Azure Kubernetes Service (AKS) nodes
- SQL Managed Instance
- Virtual Machines

These resources connect directly to your virtual network.

Virtual Network (VNet) service endpoints provide secure and direct connectivity to Azure services. These service endpoints use an optimized route over the Azure network. Service endpoints enable private IP addresses in the VNet to reach the endpoint of an Azure service without needing a public IP address on the VNet.

Private Link provides dedicated access using private IP addresses to Azure PaaS instances, or custom services behind an Azure Load Balancer Standard.

Design considerations

Network connectivity includes the following design considerations related to cost optimization:

- Running cost of services: The services are metered. Pay for service itself and consumption on service.
- VNet Peering cost: Consider the consequences of putting all resources in a single VNet to save costs. It also prevents the infrastructure from growing. The VNet can eventually reach a point where new resources don't fit anymore.
- For two peered VNets using a private endpoint: Only the private endpoint access is billed and not the VNet peering cost.
- Azure Firewall is also metered: Pay for the instance and for usage. The same applies to load balancers.

Checklist

Have you configured Network connectivity with cost optimization in mind?

- ✓ Select SKU for service so that it does the job required, which allows the customer to grow as the workload evolves.
- ✓ For the Load balancer, select two SKUs: Basic (free) and Standard (paid).
- ✓ For App Gateway, select Basic or V2.
- ✓ For Gateways, limit throughput and performance.
- ✓ Select DDoS Standard.

Configuration recommendations

Consider the following recommendation for cost optimization when configuring Network connectivity:

Recommendation	Description
For the Load balancer, select two SKUs: Basic (free) and Standard (paid).	Microsoft recommends Standard because it has richer capabilities, such as: - Outbound rules - Granular network security configuration - Monitoring Standard provides a Service Level Agreement (SLA) and can be deployed in Availability Zones. Capabilities in Basic are limited.
Select DDoS Standard.	Depending on the workload and usage patterns, Standard can provide useful protection. Otherwise, you can use Basic for small customers.

Next step

Operational excellence and Network connectivity

Operational excellence and Network connectivity

Article • 11/30/2022

Network connectivity includes three Azure models for private network connectivity:

- VNet injection
- VNet service endpoints
- Private Link

VNet injection applies to services that are deployed specifically for you, such as:

- Azure Kubernetes Service (AKS) nodes
- SQL Managed Instance
- Virtual Machines

These resources connect directly to your virtual network.

Virtual Network (VNet) service endpoints provide secure and direct connectivity to Azure services. These service endpoints use an optimized route over the Azure network. Service endpoints enable private IP addresses in the VNet to reach the endpoint of an Azure service without needing a public IP address on the VNet.

Private Link provides dedicated access using private IP addresses to Azure PaaS instances, or custom services behind an Azure Load Balancer Standard.

Design considerations

Network connectivity includes the following design considerations related to operational excellence:

- Use Private Link, where available, for shared Azure PaaS services. Private Link is generally available for several services and is in public preview for numerous ones.

- Access Azure PaaS services from on-premises through ExpressRoute private peering.

- Use either virtual network injection for dedicated Azure services or Azure Private Link for available shared Azure services. To access Azure PaaS services from on-premises when virtual network injection or Private Link isn't available, use ExpressRoute with Microsoft peering. This method avoids transiting over the public internet.

- Use virtual network service endpoints to secure access to Azure PaaS services from within your virtual network. Use virtual network service endpoints only when Private Link isn't available and there are no concerns with unauthorized movement of data.

- Service Endpoints don't allow a PaaS service to be accessed from on-premises networks. Private Endpoints do.

- To address concerns about unauthorized movement of data with service endpoints, use network-virtual appliance (NVA) filtering. You can also use virtual network service endpoint policies for Azure Storage.

- The following native network security services are fully managed services. Customers don't incur the operational and management costs associated with infrastructure deployments, which can become complex at scale:
  - Azure Firewall
  - Application Gateway
  - Azure Front Door

- PaaS services are typically accessed over public endpoints. The Azure platform provides capabilities to secure these endpoints or make them entirely private.

- You can also use third-party network-virtual appliances (NVAs) if the customer prefers them for situations where native services don't satisfy specific requirements.

Checklist

Have you configured Network connectivity with operational excellence in mind?

✓ Don't implement forced tunneling to enable communication from Azure to Azure resources.
✓ Unless you use network virtual appliance (NVA) filtering, don't use virtual network service endpoints when there are concerns about unauthorized movement of data.
✓ Don't enable virtual network service endpoints by default on all subnets.

Next step

Reliability and Azure Virtual Network

Reliability and Azure Virtual Network

Article • 11/30/2022

A fundamental building block for your private network, Azure Virtual Network enables Azure resources to securely communicate with each other, the internet, and on-premises networks.

Key features of Azure Virtual Network include:

- Communication with Azure resources
- Communication with the internet
- Communication with on-premises resources
- Network traffic filtering

For more information, reference What is Azure Virtual Network?

To understand how Azure Virtual Network supports a reliable workload, reference the following topics:

- Tutorial: Move Azure VMs across regions
- Quickstart: Create a virtual network using the Azure portal
- Virtual Network – Business Continuity

Design considerations

The Virtual Network (VNet) includes the following design considerations for a reliable Azure workload:

- Overlapping IP address spaces across on-premises and Azure regions creates major contention challenges.
- While a Virtual Network address space can be added after creation, this process requires an outage if the Virtual Network is already connected to another Virtual Network through peering. An outage is necessary because the Virtual Network peering is deleted and re-created.
- Resizing of peered Virtual Networks is in public preview (August 20, 2021).
- Some Azure services do require dedicated subnets, such as:
  - Azure Firewall
  - Azure Bastion
  - Virtual Network Gateway
- Subnets can be delegated to certain services to create instances of that service within the subnet.

- Azure reserves five IP addresses within each subnet, which should be factored in when sizing Virtual Networks and encompassed subnets.

Checklist

Have you configured Azure Virtual Network with reliability in mind?

- ✓ Use Azure DDoS Standard Protection Plans to protect all public endpoints hosted within customer Virtual Networks.
- ✓ Enterprise customers must plan for IP addressing in Azure to ensure there's no overlapping IP address space across considered on-premises locations and Azure regions.
- ✓ Use IP addresses from the address allocation for private internets (Request for Comment (RFC) 1918).
- ✓ For environments with limited private IP addresses (RFC 1918) availability, consider using IPv6.
- ✓ Don't create unnecessarily large Virtual Networks (for example: `/16`) to ensure there's no unnecessary waste of IP address space.
- ✓ Don't create Virtual Networks without planning the required address space in advance.
- ✓ Don't use public IP addresses for Virtual Networks, especially if the public IP addresses don't belong to the customer.
- ✓ Use VNet Service Endpoints to secure access to Azure Platform as a Service (PaaS) services from within a customer VNet.
- ✓ To address data exfiltration concerns with Service Endpoints, use Network Virtual Appliance (NVA) filtering and VNet Service Endpoint Policies for Azure Storage.
- ✓ Don't implement forced tunneling to enable communication from Azure to Azure resources.
- ✓ Access Azure PaaS services from on-premises through ExpressRoute Private Peering.
- ✓ To access Azure PaaS services from on-premises networks when VNet injection or Private Link aren't available, use ExpressRoute with Microsoft Peering when there are no data exfiltration concerns.
- ✓ Don't replicate on-premises perimeter network (also known as DMZ, demilitarized zone, and screened subnet) concepts and architectures into Azure.
- ✓ Ensure the communication between Azure PaaS services that have been injected into a Virtual Network is locked down within the Virtual Network using user-defined routes (UDRs) and network security groups (NSGs).
- ✓ Don't use VNet Service Endpoints when there are data exfiltration concerns, unless NVA filtering is used.
- ✓ Don't enable VNet Service Endpoints by default on all subnets.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring an Azure Virtual Network:

Recommendation	Description
Don't create Virtual Networks without planning the required address space in advance.	Adding address space will cause an outage once a Virtual Network is connected through Virtual Network peering.
Use VNet Service Endpoints to secure access to Azure Platform as a Service (PaaS) services from within a customer VNet.	Only when Private Link isn't available and when there are no data exfiltration concerns.
Access Azure PaaS services from on-premises through ExpressRoute Private Peering.	Use either VNet injection for dedicated Azure services or Azure Private Link for available shared Azure services.
To access Azure PaaS services from on-premises networks when VNet injection or Private Link aren't available, use ExpressRoute with Microsoft Peering when there are no data exfiltration concerns.	Avoids transit over the public internet.
Don't replicate on-premises perimeter network (also known as DMZ, demilitarized zone, and screened subnet) concepts and architectures into Azure.	Customers can get similar security capabilities in Azure as on-premises, but the implementation and architecture will need to be adapted to the cloud.
Ensure the communication between Azure PaaS services that have been injected into a Virtual Network is locked down within the Virtual Network using user-defined routes (UDRs) and network security groups (NSGs).	Azure PaaS services that have been injected into a Virtual Network still perform management plane operations using public IP addresses.

Next step

Operational excellence and Azure Virtual Network

Operational excellence and Azure Virtual Network

Article • 11/30/2022

A fundamental building block for your private network, Azure Virtual Network enables Azure resources to securely communicate with each other, the internet, and on-premises networks.

Key features of Azure Virtual Network include:

- Communication with Azure resources
- Communication with the internet
- Communication with on-premises resources
- Network traffic filtering

For more information, reference What is Azure Virtual Network?

To understand how Azure Virtual Network supports operational excellence, reference the following topics:

- Monitoring Azure Virtual Network
- Monitoring Azure Virtual Network data reference
- Azure Virtual Network concepts and best practices

Design considerations

The Virtual Network (VNet) includes the following design considerations for operational excellence:

- Overlapping IP address spaces across on-premises and Azure regions creates major contention challenges.
- While a Virtual Network address space can be added after creation, this process requires an outage if the Virtual Network is already connected to another Virtual Network through peering. An outage is necessary because the Virtual Network peering is deleted and re-created.
- Resizing of peered Virtual Networks is in public preview (August 20, 2021).
- Some Azure services do require dedicated subnets, such as:
    - Azure Firewall
    - Azure Bastion
    - Virtual Network Gateway

- Subnets can be delegated to certain services to create instances of that service within the subnet.
- Azure reserves five IP addresses within each subnet, which should be factored in when sizing Virtual Networks and encompassed subnets.

Checklist

Have you configured Azure Virtual Network with operational excellence in mind?

- ✓ Use Azure DDoS Standard Protection Plans to protect all public endpoints hosted within customer Virtual Networks.
- ✓ Enterprise customers must plan for IP addressing in Azure to ensure there's no overlapping IP address space across considered on-premises locations and Azure regions.
- ✓ Use IP addresses from the address allocation for private internets (Request for Comment (RFC) 1918).
- ✓ For environments with limited private IP addresses (RFC 1918) availability, consider using IPv6.
- ✓ Don't create unnecessarily large Virtual Networks (for example: `/16`) to ensure there's no unnecessary waste of IP address space.
- ✓ Don't create Virtual Networks without planning the required address space in advance.
- ✓ Don't use public IP addresses for Virtual Networks, especially if the public IP addresses don't belong to the customer.
- ✓ Use VNet Service Endpoints to secure access to Azure Platform as a Service (PaaS) services from within a customer VNet.
- ✓ To address data exfiltration concerns with Service Endpoints, use Network Virtual Appliance (NVA) filtering and VNet Service Endpoint Policies for Azure Storage.
- ✓ Don't implement forced tunneling to enable communication from Azure to Azure resources.
- ✓ Access Azure PaaS services from on-premises through ExpressRoute Private Peering.
- ✓ To access Azure PaaS services from on-premises networks when VNet injection or Private Link aren't available, use ExpressRoute with Microsoft Peering when there are no data exfiltration concerns.
- ✓ Don't replicate on-premises perimeter network (also known as DMZ, demilitarized zone, and screened subnet) concepts and architectures into Azure.
- ✓ Ensure the communication between Azure PaaS services that have been injected into a Virtual Network is locked down within the Virtual Network using user-defined routes (UDRs) and network security groups (NSGs).

- ✓ Don't use VNet Service Endpoints when there are data exfiltration concerns, unless NVA filtering is used.
- ✓ Don't enable VNet Service Endpoints by default on all subnets.

Configuration recommendations

Consider the following recommendations for operational excellence when configuring an Azure Virtual Network:

Recommendation	Description
Don't create Virtual Networks without planning the required address space in advance.	Adding address space will cause an outage once a Virtual Network is connected through Virtual Network peering.
Use VNet Service Endpoints to secure access to Azure Platform as a Service (PaaS) services from within a customer VNet.	Only when Private Link isn't available and when there are no data exfiltration concerns.
Access Azure PaaS services from on-premises through ExpressRoute Private Peering.	Use either VNet injection for dedicated Azure services or Azure Private Link for available shared Azure services.
To access Azure PaaS services from on-premises networks when VNet injection or Private Link aren't available, use ExpressRoute with Microsoft Peering when there are no data exfiltration concerns.	Avoids transit over the public internet.
Don't replicate on-premises perimeter network (also known as DMZ, demilitarized zone, and screened subnet) concepts and architectures into Azure.	Customers can get similar security capabilities in Azure as on-premises, but the implementation and architecture will need to be adapted to the cloud.
Ensure the communication between Azure PaaS services that have been injected into a Virtual Network is locked down within the Virtual Network using user-defined routes (UDRs) and network security groups (NSGs).	Azure PaaS services that have been injected into a Virtual Network still perform management plane operations using public IP addresses.

Next step

Reliability and ExpressRoute

Reliability and Azure Load Balancer

Article • 11/30/2022

Load balancing refers to evenly distributing load (incoming network traffic) across a group of backend resources or servers. With Azure Load Balancer, load-balance traffic to and from virtual machines and cloud resources, and in cross-premises virtual networks.

You can scale your applications and create highly available services with Azure Load Balancer. It supports both inbound and outbound scenarios. Load balancer provides low latency and high throughput.

Key benefits include:

- Load balance internal and external traffic to Azure virtual machines.
- Increase availability by distributing resources within and across zones.
- Configure outbound connectivity for Azure virtual machines.
- Use health probes to monitor load-balanced resources.

For more information, reference Why use Azure Load Balancer?

To understand how Azure Load Balancer supports a reliable workload, reference the following topics:

- Improve application scalability and resiliency by using Azure Load Balancer
- Load Balancer and Availability Zones
- High availability ports overview

Checklist

Have you configured Azure Load Balancer with reliability in mind?

✓ For production workloads, use the Standard Stock Keeping Units (SKU).

Configuration recommendations

Consider the following recommendation to optimize reliability when configuring an Azure Load Balancer:

Recommendation	Description
For production workloads, use the Standard Stock Keeping Units (SKU).	Basic load balancers don't have a Service Level Agreement (SLA). The Standard SKU supports Availability Zones.

Next step

Operational excellence and Azure Load Balancer

Operational excellence and Azure Load Balancer

Article • 03/01/2023

Load balancing refers to evenly distributing load (incoming network traffic) across a group of backend resources or servers. With Azure Load Balancer, load-balance traffic to and from virtual machines and cloud resources, and in cross-premises virtual networks.

You can scale your applications and create highly available services with Azure Load Balancer. It supports both inbound and outbound scenarios. Load balancer provides low latency and high throughput.

Key benefits include:

- Load balance internal and external traffic to Azure virtual machines.
- Increase availability by distributing resources within/across Azure regions and zones.
- Configure outbound connectivity for Azure virtual machines.
- Use health probes to monitor load-balanced resources.

For more information, reference Why use Azure Load Balancer?

To understand how Azure Load Balancer supports operational excellence, reference the following topics:

- Load Balancer health probes
- Standard load balancer diagnostics with metrics, alerts, and resource health
- Using Insights to monitor and configure your Azure Load Balancer

Checklist

Have you configured Azure Load Balancer with operational excellence in mind?

✓ For production workloads, use the Standard Stock Keeping Units (SKU).

Configuration recommendations

Consider the following recommendation for operational excellence when configuring an Azure Load Balancer:

Recommendation	Description

Recommendation	Description
For production workloads, use the Standard Stock Keeping Units (SKU).	Basic load balancers don't have a Service Level Agreement (SLA). The Standard SKU supports Availability Zones and multi-region load balancing.

Next step

Reliability and Traffic Manager

Reliability and Traffic Manager

Article • 11/30/2022

Traffic Manager is a Domain Name System (DNS)-based traffic load balancer. This service allows you to distribute traffic to your public-facing applications across the global Azure regions. Traffic Manager also provides your public endpoints with high availability and quick responsiveness.

Features include:

- Increase application availability
- Improve application performance
- Service maintenance without downtime
- Combine hybrid applications
- Distribute traffic for complex deployments

For more information, reference What is Traffic Manager?

To learn how Traffic Manager supports a reliable workload, reference the following articles:

- Enhance your service availability and data locality by using Azure Traffic Manager
- Using load-balancing services in Azure
- Disaster recovery using Azure DNS and Traffic Manager

Checklist

Have you configured Traffic Manager with reliability in mind?

- ✓ If the Time to Live (TTL) interval of the DNS record is too long, consider adjusting the health probe timing or DNS record TTL.
- ✓ Implement a custom page to use as a health check for your Traffic Manager.
- ✓ Evaluate the three different traffic routing methods.
- ✓ Consider nested Traffic Manager profiles.

Configuration recommendations

Consider the following recommendations to optimize reliability when configuring Traffic Manager:

Recommendation	Description

Recommendation	Description
If the Time to Live (TTL) interval of the DNS record is too long, consider adjusting the health probe timing or DNS record TTL.	When a backend becomes unavailable, Traffic Manager won't fail over to another region immediately. There will be a time interval where clients can't be served. The length of this interval depends on the time settings of the health probe (probe interval and the number of unhealthy responses allowed). If the resulting interval is still too large for the scenario, consider switching to Azure Front Door for global load balancing.
Implement a custom page to use as a health check for your Traffic Manager.	A common practice is to implement a custom page within your application (for example: `/health.aspx`). Using this path for monitoring, you can do application-specific checks, such as checking performance counters or verifying database availability. Based on these custom checks, the page returns an appropriate `HTTPS` status code.
Evaluate the three different traffic routing methods.	Traffic Manager supports three traffic-routing methods to determine how to route network traffic to the various service endpoints. Traffic Manager applies the traffic-routing method to each DNS query it receives. The traffic-routing method determines which endpoint is returned in the DNS response. The customer should be aware of these endpoints and the differences in routing between endpoints.
Consider nested Traffic Manager profiles.	Each Traffic Manager profile specifies a single traffic-routing method. There are scenarios that require more sophisticated traffic routing than the routing provided by a single Traffic Manager profile. You can nest Traffic Manager profiles to combine the benefits of more than one traffic-routing method. Nested profiles allow you to override the default Traffic Manager behavior to support larger, more complex application deployments.

Next step

Operational excellence and Traffic Manager

Operational excellence and Traffic Manager

Article • 11/30/2022

Traffic Manager is a Domain Name System (DNS)-based traffic load balancer. This service allows you to distribute traffic to your public-facing applications across the global Azure regions. Traffic Manager also provides your public endpoints with high availability and quick responsiveness.

Features include:

- Increase application availability
- Improve application performance
- Service maintenance without downtime
- Combine hybrid applications
- Distribute traffic for complex deployments

For more information, reference What is Traffic Manager?

To learn how Traffic Manager supports operational excellence, reference the following articles:

- Troubleshooting degraded state on Azure Traffic Manager
- Traffic Manager endpoint monitoring
- Traffic Manager metrics and alerts

Checklist

Have you configured Traffic Manager with operational excellence in mind?

✓ If the Time to Live (TTL) interval of the DNS record is too long, consider adjusting the health probe timing or DNS record TTL.

Configuration recommendations

Consider the following recommendation for operational excellence when configuring Traffic Manager:

Recommendation	Description

Recommendation	Description
If the Time to Live (TTL) interval of the DNS record is too long, consider adjusting the health probe timing or DNS record TTL.	When a backend becomes unavailable, Traffic Manager won't fail over to another region immediately. There will be a time interval where clients can't be served. The length of this interval depends on the time settings of the health probe (probe interval and the number of unhealthy responses allowed). If the resulting interval is still too large for the scenario, consider switching to Azure Front Door for global load balancing.

Next step

Cost optimization and IP addresses

Cost optimization and IP addresses

Article • 11/30/2022

IP services are a collection of IP address-related services that enable communication in an Azure Virtual Network. Public and private IP addresses are used in Azure for communication between resources. The communication with resources can occur in a private Azure Virtual Network and the public internet.

Key features include:

- Public IP addresses
- Public IP address prefixes
- Private IP addresses
- Routing preference
- Routing preference unmetered

For more information, reference What is Azure Virtual Network IP Services?

To understand how IP services support a cost-optimized workload, reference the following articles:

- IP addresses pricing
- Create, change, or delete an Azure public IP address
- Routing over public Internet (ISP network)

Checklist

Have you configured IP addresses with cost optimization in mind?

✓ PIPs (Public IPs) are free until used. Static PIPs are paid even when not assigned to resources.

Configuration recommendations

Consider the following recommendation for cost optimization when configuring IP addresses:

Recommendation	Description

Recommendation	Description
PIPs (Public IPs) are free until used. Static PIPs are paid even when not assigned to resources.	There's a difference in billing for regular and static public IP addresses. Develop a process to look for orphan network interface cards (NICs) and PIPs that aren't being used in production and non-production.

Next step

Cost optimization and Log Analytics

Cost optimization and Log Analytics

Article • 03/24/2023

Log Analytics is a tool in the Azure portal used to edit and run log queries with data in Azure Monitor Logs. You can write a query that returns a set of records. Then, use features of Log Analytics to sort, filter, and analyze them. Alternately, write a more advanced query to do statistical analysis. Visualize the results in a chart to identify a particular trend.

For more information, reference Overview of Log Analytics in Azure Monitor.

To understand how Log Analytics supports cost optimization, reference Manage usage and costs with Azure Monitor Logs. This article includes:

- Estimating the costs to manage your environment
- Viewing Log Analytics usage on your Azure bill
- Understand your usage and optimizing your pricing tier

Design considerations

Log Analytics workspace includes the following design considerations:

- Consider how long to retain data on Log Analytics:

  Data ingested into Log Analytics workspace can be retained at no additional charge up to the first `31` days. Consider general aspects to configure the Log Analytics workspace level default retention. Consider specific needs to configure data retention by data type that can be as low as four days. Example: Usually, performance data doesn't need to be retained longer, instead, security logs may need to be retained longer.

- Consider exporting data for long-term retention or auditing purposes:

  Data retained for audit purposes may be exported to a cheaper storage type. Refer to Log Analytics workspace data export in Azure Monitor (preview).

Checklist

Have you configured Log Analytics with cost optimization in mind?

✓ Consider adoption of the Commitment Tiers pricing model to the Log Analytics workspace.

- ✓ Evaluate usage of daily cap to limit the daily ingestion for your workspace.
- ✓ Understand Log Analytics workspace usage.
- ✓ Evaluate possible data ingestion volume reducing.

Configuration recommendations

Consider the following recommendation for cost optimization when configuring Log Analytics:

Recommendation	Description
Consider adoption of the Commitment Tiers pricing model to the Log Analytics workspace.	The usage of Commitment Tiers enables saving as much as `30%` compared to Pay-As-You-Go pricing. Commitment Tiers starts at `100 GB/day` and any usage above the reservation level is billed at the Pay-As-You-Go rate. Refer to Changing pricing tier about how to change the pricing tier to Capacity Reservations. Use the Log Analytics usage and estimated cost page to analyze data usage and calculate possible Commitment Tiers. *Note: Azure Defender (Security Center) billing includes* `500 MB/node/day` *allocation against the security data types. Take it into consideration when calculating Commitment Tiers.*
Evaluate daily cap usage to limit the daily ingestion for your workspace.	Daily cap is used to manage an unexpected increase in data volume. Use daily cap when you want to limit unplanned charges for your workspace. Use care with this configuration as it can cause some data to be unwritten on Log Analytics workspace if the daily cap is reached. This configuration can impact services whose functionality may depend on the availability of up-to-date data in the workspace. Refer to Set the Daily Cap about how to set the daily cap in Application Insights. *Note: If you have a workspace-based Application Insights, use daily cap in workspace to limit ingestion and costs instead of using the cap in Application Insights.*
Understand Log Analytics workspace usage.	When Log Analytics workspace usage is higher than expected, consider the troubleshooting guide and the Understanding ingested data volume guide to understand the unexpected behavior.
Evaluate possible data ingestion volume reducing.	Refer to Tips for reducing data volume documentation to help configure data ingestion properly.

Next step

Security and Application Insights

Security and Application Insights

Article • 03/24/2023

Application Insights is a feature of Azure Monitor. This feature provides extensible application performance management (APM) and monitoring for live web apps.

Key features include:

- Supports a wide variety of platforms, including .NET, Node.js, Java, and Python.
- Works for apps hosted on-premises, hybrid, or on any public cloud.
- Integrates with DevOps processes.
- Has connection points to many development tools.
- Can monitor and analyze customer data from mobile apps by integrating with Visual Studio App Center.

For more information, reference Application Insights overview.

Checklist

Have you configured Application Insights with security in mind?

✓ Review instances where customer data is captured in your application.

Configuration recommendations

Consider the following security recommendation when configuring Application Insights:

Recommendation	Description
Review instances where customer data is captured in your application.	We don't recommend collecting customer data in Application Insights, although it can be unavoidable. It's up to you and your company to determine the strategy you'll use to handle your private data.

Next step

Cost optimization and Application Insights

Cost optimization and Application Insights

Article • 03/24/2023

Application Insights is a feature of Azure Monitor. This feature provides extensible application performance management (APM) and monitoring for live web apps.

Key features include:

- Supports a wide variety of platforms, including .NET, Node.js, Java, and Python.
- Works for apps hosted on-premises, hybrid, or on any public cloud.
- Integrates with DevOps processes.
- Has connection points to many development tools.
- Can monitor and analyze customer data from mobile apps by integrating with Visual Studio App Center.

For more information, reference Application Insights overview.

Design considerations

Application Insights includes the following design considerations for cost optimization:

- Consider using sampling to reduce the amount of data that's sent:

  Sampling is a feature in Application Insights. It's a recommended way to reduce data traffic, data, and storage costs. Refer to Sampling in Application Insights.

- Consider turning off collection for unneeded modules:

  On configuration files, you can enable or disable data modules and initializers for tracking data from your applications. Refer to Application Insights for web pages.

- Consider limiting Asynchronous JavaScript and XML (AJAX) call tracing:

  AJAX calls can be limited to reduce costs. Refer to Application Insights for web pages, which explains the fields and its configurations.

Checklist

Have you configured Application Insights with cost optimization in mind?

✓ Evaluate usage of daily cap to limit the daily ingestion for your workspace.

✓ Use sampling in Azure Application Insights to reduce data traffic, data costs, and storage costs, while preserving a statistically correct analysis of application data.

Configuration recommendations

Consider the following recommendations for cost optimization when configuring Application Insights:

Recommendation	Description
Evaluate daily cap usage to limit the daily ingestion for your workspace.	Daily cap is used to manage an unexpected increase in data volume. Use daily cap when you want to limit unplanned charges for your workspace. Use care with this configuration as it can cause some data to be unwritten on Log Analytics workspace if the daily cap is reached. This configuration can impact services whose functionality may depend on the availability of up-to-date data in the workspace. Refer to Set the Daily Cap about how to set the daily cap in Application Insights. *Note: If you have a workspace-based Application Insights, use the daily cap in workspace to limit ingestion and costs instead of using the cap in Application Insights.*

Next step

Operational excellence and Application Insights

Operational excellence and Application Insights

Article • 03/24/2023

Application Insights is a feature of Azure Monitor. This feature provides extensible application performance management (APM) and monitoring for live web apps.

Key features include:

- Supports a wide variety of platforms, including .NET, Node.js, Java, and Python.
- Works for apps hosted on-premises, hybrid, or on any public cloud.
- Integrates with DevOps processes.
- Has connection points to many development tools.
- Can monitor and analyze customer data from mobile apps by integrating with Visual Studio App Center.

For more information, reference Application Insights overview.

Checklist

Have you configured Application Insights with operational excellence in mind?

- ✓ Configure Application Insights to monitor the availability and responsiveness of your web application.
- ✓ Be aware that Application Insights can be used to monitor deployed sites and services on-premises (or on an Azure Virtual Machine (VM)).
- ✓ Evaluate Java codeless application monitoring for your Java-based application development stack.
- ✓ Configure sampling in Application Insights.
- ✓ Record custom events and metrics from sites and services in Application Insights.
- ✓ Use Application Insights to ingest existing log traces from common libraries, such as `ILogger`, `Nlog`, and `log4Net`.
- ✓ Become familiar with the Application Insights quotas and limits.
- ✓ Review the need for custom analysis. Use Application Insights data with tools such as Azure Dashboards or Power BI.
- ✓ Separate data across Application Insights resources.

Configuration recommendations

Consider the following recommendations for operational excellence when configuring Application Insights:

Recommendation	Description
Configure Application Insights to monitor the availability and responsiveness of your web application.	After you've deployed your application, you can set up recurring tests to monitor availability and responsiveness. Application Insights sends web requests to your application at regular intervals from points around the world. It can alert you if your application isn't responding or if it responds too slowly.
Evaluate Java codeless application monitoring for your Java-based application development stack.	Java codeless application monitoring is all about simplicity. There are no code changes. You can enable the Java agent through a couple of configuration changes. The Java agent works in any environment and allows you to monitor all your Java applications. No matter if you're running your Java apps on Virtual Machines, on-premises, in Azure Kubernetes Service (AKS), on Windows, or Linux, the Java `3.0` agent will monitor your app.
Configure sampling in Application Insights.	Ingestion sampling operates at the point where the data from your web servers, browsers, and devices reaches the Application Insights service endpoints. Although it doesn't reduce the data sent from your app, it does reduce the amount processed, retained, and charged by Application Insights. Use this type of sampling if your app often goes above its monthly quota. Use ingestion sampling if you don't have access to the Software Development Kit (SDK)-based types of sampling.
Record custom events and metrics from sites and services in Application Insights.	Use Application Insights to record domain-specific custom events and metrics from your site or service. For example: *number-of-active-baskets* or *product-lines-out-of-stock*.
Use Application Insights to ingest existing log traces from common libraries, such as `ILogger`, `Nlog`, and `log4Net`.	If you're already using a logging framework such as `ILogger`, `Nlog`, `log4Net`, or `System.Diagnostics.Trace`, we recommend sending your diagnostic tracing logs to Application Insights. For Python applications, send diagnostic tracing logs using `AzureLogHandler` in OpenCensus Python for Azure Monitor. You can explore and search these logs, which are merged with the other log files from your application. Merging the log files allows you to identify traces associated with each user request and correlate them with other events and exception reports.

Recommendation	Description
Become familiar with the Application Insights quotas and limits.	This information can influence your sampling model and your strategy for separating Application Insights resources.
Review the need for custom analysis. Use Application Insights data with tools such as Azure Dashboards or Power BI.	There are several available options to analyze your Application Insights data. For example, you can create a dashboard in the Azure portal that includes tiles visualizing data from multiple Azure resources across different resource groups and subscriptions. Alternatively, you can use Power BI to analyze data combined with data from other sources and share insights.
Separate data across Application Insights resources.	It's important to consider when to share a single Application Insights resource and when to create a new one. For example, you should use a single resource for application components that you deploy together, a single Team develops, or that the same set of DevOps or ITOps users manages. You should use a separate resource for different environments.

Next step

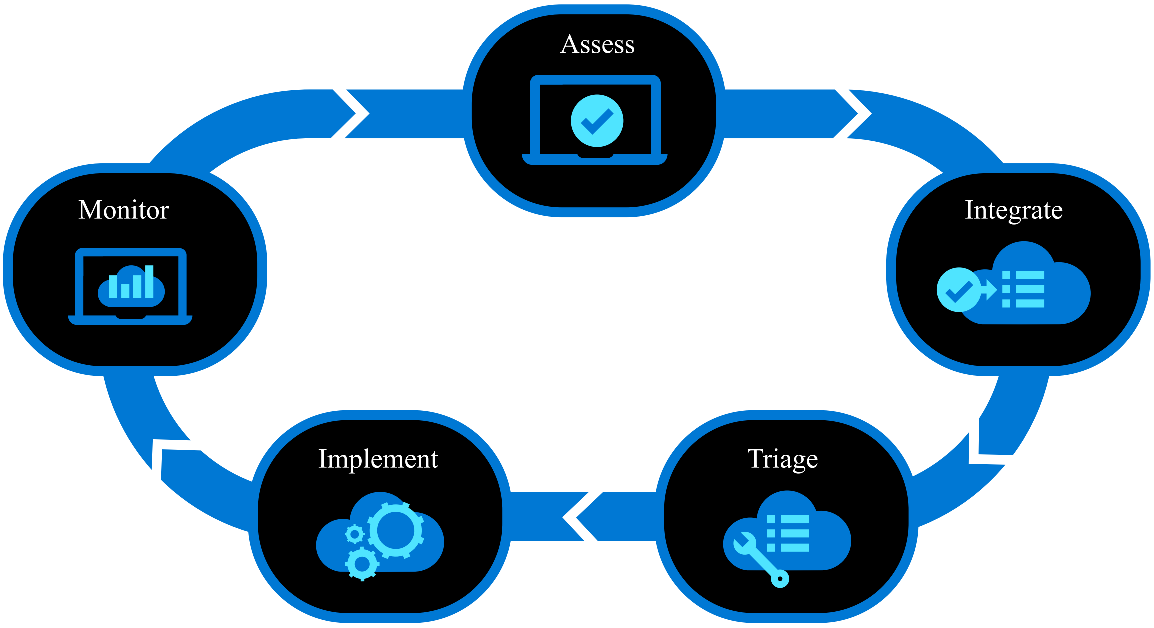
Operational excellence and Application Insights

Implementing recommendations

Article • 11/30/2022

Overview

The purpose of this document is to provide a guide for incorporating the recommendations generated by the Microsoft Azure Well-Architected Review and Azure Advisor into a new or established operational process for continuous workload improvement.



Assess workload

This first step in the Well-Architected Recommendation Process is to conduct an assessment of your workload. The Microsoft Azure Well-Architected Review tool generates a set of recommendations through a guided assessment based on the Microsoft Well-Architected Framework. This tool also has the ability to pull in Azure Advisor recommendations based on an Azure subscription or resource group. At the end of the assessment, there is an option to export these recommendations into a CSV file that can then be used to incorporate them into the operational process for the workload.

ⓘ Note

When using the **Microsoft Azure Well-Architected Review** tool, it's important to **Sign in** and select the Azure subscription or resource group that contains your workload. This will ensure that only the relevant **Azure Advisor** recommendations are included when exporting the CSV.

Integrate recommendations

The goal of this stage is to establish a backlog of work items either through a manual or an automated process that is based on the recommendations generated by the assessment. The process and tooling for managing the backlog for your workload should be well-defined based on your cloud operating model.

ⓘ **Note**

The **Microsoft Cloud Adoption Framework for Azure** is proven guidance that's designed to help you create and implement the business and technology strategies necessary for your organization to succeed in the cloud. This guidance can help you define the **cloud operating model** that governs the operational process for your workload.

If you're using a DevOps approach for your operational process, Microsoft has provided example scripts that will automate the import of the recommendations from the Well-Architected Review exported CSV into a new Azure DevOps or GitHub project within your existing organization.

Download example import scripts at DevOps Tooling for Well-Architected Recommendation Process.

Triage backlog

Now that the recommendations have been added to the backlog, the workload owners and key stakeholders should prioritize, then triage the recommendations in a weekly standup meeting with the workload team. Next, they assign recommendations to a specific owner, postpone, or dismiss. When assigned to a specific owner, the recommendation should be tracked until resolved and weekly, or monthly reminders are sent to the assignee.

When going through this process, it's recommended to align responsibilities across teams by developing a cross-team matrix that identifies *responsible, accountable, consulted, and informed (RACI)* parties. Some of the key benefits of this exercise include:

- Assisting teams in charting roles and responsibilities in a consistent manner
- Assisting teams with development of implementation tool kits
- Clarifying individual and departmental roles, and responsibilities
- Identifying accountabilities
- Eliminating misunderstandings and encouraging teamwork
- Reducing duplication of effort
- Establishing *consults* and *informs* resulting in better communication

Implement work items

This stage of the process is focused on working through the backlog of recommendations. How you organize this work will depend on your cloud operating model. For example, an Agile-based process would have an emphasis on sprint planning paired with frequent stand ups so that progress can be tracked and reported.

Monitor progress

This stage of the process is to monitor improvements to a workload relative to the Microsoft Well-Architected Framework through each iteration. It's important to establish a baseline based on the Microsoft Azure Well-Architected Review and Azure Advisor so that you can track improvements back to key recommendations and metrics.

The Azure Advisor Score is a key metric that is available for tracking and socializing improvements during this continuous process.