

where we can optimize our system, while prioritizing tenant security and providing an outstanding customer experience.

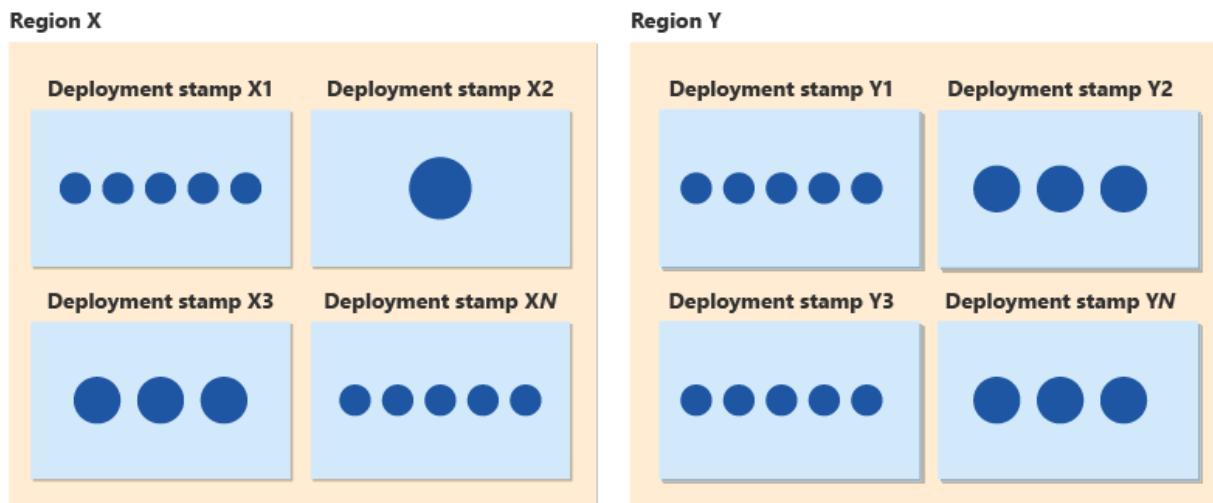
Deployment stamps

Dynamics 365 operates at hyperscale. There are hundreds of thousands of customers, with millions of users, each depending on our products. These numbers continue to grow over time. SaaS solutions typically need to be architected to scale and need to support customers across the planet.

In the cloud, it's critical to move from *scaling up* to *scaling out* wherever possible. If additional demand can be met by adding more nodes (scaling out) instead of making existing nodes more powerful (scaling up), and that relationship is close to linear, then an approach based on scale-out provides the potential to drive even higher scale.

Dynamics 365 uses a scale-out model at the application tier. Integrated monitoring detects increases in load for specific tenants and adds more nodes to meet the demand.

In conjunction with your tenant model and scale-out architecture, you can follow the [Deployment Stamps pattern](#), with each stamp supporting a set of customers. When a stamp approaches its maximum capacity, you can provision a new stamp and start to deploy new customers there. By using stamps, you can support continued customer growth, and you can expand your regional presence to new geographies.



By using deployment stamps, you also gain reliability benefits. You can roll out our updates progressively, and safe deployment processes help you to gradually roll changes out across a global fleet. Each stamp is independent of others, so if a stamp experiences a problem, only the subset of customers allocated to that stamp are affected. Stamps help you to reduce the *blast radius* of a problem or fault and contribute to an overall disaster recovery strategy.

As with every architectural decision, base your use of deployment stamps on your business needs. Deploying a stamp requires deploying a set of infrastructure to support it. If the minimum size of a stamp is too large, it's difficult to justify deploying a new stamp into a new market because you need to reach a critical mass of customers first. It's also important to understand how your customers' growth affects their use of the product because as they grow, they use more of the stamp's resources. These considerations are as important for a small ISV as they are for a hyperscale platform like Dynamics 365.

Control planes and configuration

When an ISV moves to a cloud-based SaaS delivery model, one of the most dramatic changes is that they take responsibility for the operation of the service. In most on-premises software, customers' IT departments are responsible for deploying, configuring, and managing the systems. Customers themselves take care of monitoring systems and make decisions about when to roll out updates. They're also responsible for executing all the steps involved. Often, specialist service integration partners help customers to operate complex products in their environment. The software provider becomes responsible for all these activities *across all their customers* by moving to a cloud and SaaS model. With the transition to SaaS, it's necessary to build the ISV's service and also a *control plane* to automate the work of onboarding and managing tenants. Control planes and automation are important, even with a relatively small number of customers.

It's good practice to design a control plane that's resilient, reliable, and highly available. Too often, control planes are treated as an afterthought in the journey to building a SaaS product. But if a control plane isn't designed with the same care as the rest of the product, you're at risk of it being a single point of failure. Without proper attention to the resiliency of the control plane, a control plane failure could affect all customers.

In Dynamics 365, we have a service-level control plane, which handles operations like onboarding new tenants. We also have a tenant-level control plane, which enables a customer's administration team to initiate maintenance activities and change configurations themselves because they can perform these operations through the service.

Customization and extensibility

A core value proposition of the SaaS model is that all customers run one version of the service code. When customers run one version of the service code, issues are identified and fixed once, and all customers get the benefit of those solutions quickly. The goal is

to be able to continuously evolve the one version of the service without customers having to plan for testing and deployment of updates.

To achieve this benefit, there are many changes required compared to running software in the on-premises world. For example, you need to plan processes and procedures to reduce the likelihood of regressions.

In the Dynamics 365 transformation, one area we invested in was the development of rich model-driven extensibility. ERP applications demand extensibility to support integration with other critical business systems and to meet the unique functional needs of specific customers. Instead of customization at a source code level, which was typical with on-premises applications, we introduced capabilities to extend data models through tenant-specific metadata and to trigger extended logic based on events that occur in the system.

We added isolation and governance capabilities to protect the service and other tenants from issues in another tenant's extended logic. Our approach gave customers the required level of extensibility but enabled them all to still run with the same product version. Additionally, updates could be delivered to the product without customers having to merge our changes and rebuild their code to make their extensions work with newer versions of the product.

Customization might not be a requirement for every product, but if a product does require it, customization becomes a critical design factor. You must meet the requirement without compromising the core benefits of the SaaS model. This requirement was a significant focus for Dynamics 365. The model-driven extensibility both preserved the SaaS value proposition and improved the ability of customers to create and maintain their extensions.

How we designed Dynamics 365 for resiliency

As you consider your deployment model on Azure, a critical component to consider is your resiliency if there are issues in a dependent service—for example, a networking issue, a power problem, or the maintenance of a virtual machine. In the on-premises world, where the infrastructure serves a single customer tenant, many customers rely on high availability strategies for each infrastructure component. But when you consider resiliency at cloud scale, high availability is often necessary but not sufficient. With enough scale, failures happen.

A core focus area for Dynamics 365 today is targeting redundancy across Azure availability zones to allow the mission-critical Dynamics services to seamlessly continue operating, even if an outage impacts a datacenter or an entire availability zone.

To apply this mindset to your own solution, there are some important practices to follow.

- Make sure that you invest in monitoring tools to quickly identify problems. With SaaS, your customers expect you to know about outages and to engage rapidly to restore service.
- Use platform capabilities like availability zones and zone redundancy if they're appropriate for your service.
- Design your applications for resiliency at every layer. For example, it's important to also consider other cloud best practices like using [retries](#), [circuit breakers](#), and [bulkheads](#), and adopting asynchronous communication practices. These practices can keep your service healthy even when other services you depend on are under stress.
- Consider the availability of your control plane, especially because it has a role in the recovery of your solution when infrastructure assets are impacted.
- When you've implemented capabilities for resiliency, run tests. You never know if your plans and features are complete until you try to use them. It can be useful to exercise your failover processes as part of your normal maintenance activities, which can give you both an approach to maintenance without downtime and a validation of your failover mechanisms.

The [reliability pillar of the Azure Well-Architected Framework](#) provides great guidance on these topics.

How we adapted to a cloud environment

Dynamics 365 has evolved into a sophisticated cloud-native architecture, but it's common for ISVs to make more limited *lift-and-shift* transitions from on-premises environments into the cloud. We discussed the model of [defining an MVP](#) to get your SaaS service into customers' hands quickly, which begins the cycle of learning and continuous improvement. But there's a balance. *Lift and shift* should really be *Lift, shift, and adapt*.

Earlier in this article, we discussed [designing for resiliency with availability zones and other cloud best practices](#). There are other areas where common on-premises design patterns lead to challenges or higher costs in the cloud, too. For example, in on-premises applications, it's common to store binary large objects in a relational database. For example, you might store a PDF document related to a sales order as part of the sales order in an SQL database. In the on-premises world, this approach simplifies consistency across backups and point-in-time restore functions. However, in the cloud, large objects stored in the database can be costly. Additionally, Azure Storage blobs

simplify storing large binary objects, with straightforward logic required to preserve consistent backups.

It's important to think about the things that you *need* to do as part of a cloud transformation. You should do those things that produce a stronger cloud product. But you should also use that as an opportunity to get to market quickly and begin the virtuous cycle of learning and continuous improvement.

The cloud can also make entirely new solutions practical that weren't an option in an on-premises environment. One of the most performance-intensive processes in an ERP system is manufacturing resource planning, or *MRP II*. MRP II looks at inventory on hand, the expected incoming and outgoing orders, and manufacturing requirements. Then it determines what a business must buy or make to satisfy expected orders. In the on-premises Dynamics, this functionality was implemented in application code that worked directly against the relational store. The planning function consumed a lot of system capacity and ran for an extended time. In the first cloud versions, the on-premises functionality was brought forward unchanged—it worked, but with the same scale and performance challenges. Then, a few years ago, we introduced a new in-memory microservice that could complete the same planning run in a fraction of the time and without the performance impact. Importantly, because the microservice is a critical core of a manufacturing system, we introduced it as a capability that customers could opt in to after verifying in their sandbox environments that it produced the correct results. As more customers pivoted to the new microservice, we triggered efforts to get every customer to use the microservice so that the old capability could be deprecated. With MRP II becoming something that could be run in minutes at any time, organizations could be nimbler. The cloud made creating and connecting an in-memory microservice practical, and good SaaS engineering principles allowed even this most critical part of the service to evolve without disrupting customers.

How we migrated existing customers to the cloud

Migration of an existing customer base can be the fastest way to grow a cloud service to scale. However, when we brought Dynamics 365 to the cloud, we focused initially on new customers. There were two key reasons:

- This approach gave us a way to gauge whether we had delivered a SaaS solution that won on its own merits and not just one that appealed to on-premises customers looking for a simple cloud migration.
- We could focus on the MVP and defer tasks like building tools for migrating existing customers.

After we saw traction with new customers, we then were able to focus on migrating existing on-premises customers.

We found that customers are often afraid of the cost and complexity of the move. It was important for us to provide tools that reduce the cost and remove the unknown factors. We developed tools to help with analyzing the effort involved in migrating their data to new schemas that had evolved in our cloud product and to understand the impact of the migration on the customer's extensions and integrations. We also found that it was helpful to build out other tools and programs that put boundaries around the time and cost to migrate.

Moving to the cloud alone benefits customers by removing much of the systems management burden they face with on-premises products, but highlighting the benefits of your cloud version is an important motivator, too.

How we learned to operate Dynamics 365 as SaaS

After you've defined an MVP and done the engineering to lift, shift, and adapt, you need to focus on operating the SaaS service on behalf of your customers. This transformation is enormous. In the on-premises world, software providers create and ship the software, system integrators deploy it, and the customer's IT organization or outsourced provider runs it. With SaaS, not only is the SaaS provider principally responsible for operating the service, but they're also responsible for operating it for hundreds to thousands of customers at the same time.

We learned a lot by operating Dynamics 365 in the cloud for a large and growing number of customers.

Monitor: As a service provider, customers expect you to detect service health issues before they do, and they expect you to immediately work on resolutions. A health issue isn't just when the service is down. A customer's view of a service being unhealthy includes the service performing slowly or behaving incorrectly. It's essential that you develop adequate monitoring tools—this development is part of your service, not an optional accessory.

Communicate: In the on-premises world, the customer can see their IT team working on a problem. In the cloud, they can't. It's essential to communicate when you detect a service health issue, to keep communicating on the progress to resolution, and to confirm the *all clear* when the issue is resolved. The nature of the communications varies with the severity of the issue. Your communications pipeline is also a core part of your

SaaS service, and you need to ensure that communications can succeed even when core parts of your SaaS service's infrastructure are compromised.

Whole-stack view: In the on-premises world, the application provider is generally responsible for the application component, and the customer owns the underlying infrastructure. In the cloud, you're responsible for the whole stack. If the service has a health issue, the customer looks to you to detect, communicate, and repair, whether the issue is in the application or in the cloud platform it runs on.

Automate: If humans are required to perform manual steps in the operation of the service, mistakes will inevitably be made. Every possible action should be automated and logged. If an action is required on enough service nodes, automation is the only option. A great example is the database administration for Dynamics 365. With our decision to keep each tenant's data in a separate Azure SQL database, we needed to develop automation to handle all the tasks typically performed by a DBA, for example, index maintenance and query optimization. For more information on how we manage databases at scale, see [Running 1M databases on Azure SQL for a large SaaS provider](#).

Safe deployment: Wherever possible, changes should follow a safe deployment process. First, changes are introduced to low-risk environments—for example, a cloud region with only smaller customers or less critical workloads. Next, they progress to a group of slightly larger, more complex customers, and so on, until all customers have been updated. At every step, there needs to be monitoring to evaluate whether the change is successful. If there's an issue, the process should stop the change rollout and mitigate issues, or roll it back where it has already been deployed. Safe deployment practices apply to both code and configuration changes. For more information, see [Advancing safe deployment practices](#).

Live-site incident management: For us, a *live-site incident* means that a customer is having an issue with our service in production that requires engineering engagement. It might be a health issue that we detect or an issue reported by the customer that our support teams aren't able to resolve on their own. Live-site excellence is critical to SaaS success. Here are a few key points from our experience:

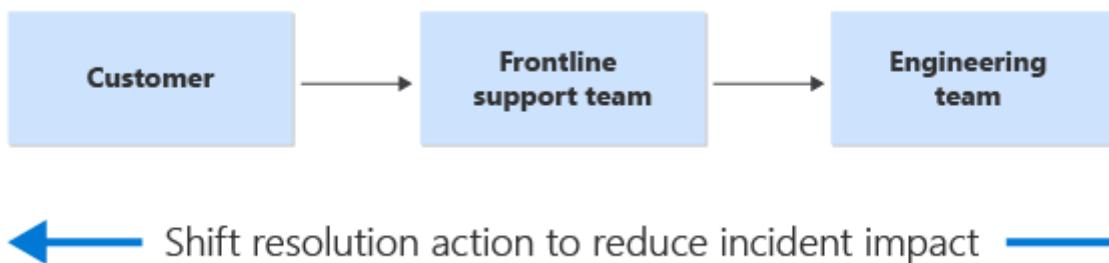
- The engineering team should handle live-site incidents. In the past, many companies had separate operations or support engineering teams. We made an explicit choice to have our core engineering teams cover live-site incidents. They have the best expertise, and seeing issues first-hand inspires the right creativity and energy to drive real, rapid improvement and better future designs. It's something that needs to be considered when planning development schedules, but it drives great results.

- Live-site incident leadership is a skill, and it's hard work—recognize it, train for it, learn to hire for it, and reward it.
- The priority should be detection, isolation, and mitigation. Get the customer healthy again, and then worry about longer term improvements.

Learn and improve: Someone once said, "Never waste a good crisis." Every live-site incident is an opportunity to improve. After mitigation is completed, make sure that you ask how to detect similar issues faster, how to correct the underlying issue to fully cure it, how to minimize the impact of similar issues, whether other similar issues might exist elsewhere in the service, and how to prevent the entire class of issues. Prioritizing these corrective actions improves service quality and reduces the demand for future live-site incidents. Service quality must improve over time, otherwise as you grow, the impact of every issue also gets higher.

Shift left: Issues that require the engagement of the live-site team are expensive. It takes time for issues to get to them, and the live-site team is a scarce resource that needs to be available for the most serious service health issues and management tasks.

Wherever possible, the best solution is eliminating an issue altogether, followed quickly by automated detection and automated mitigation. When that's not possible, *shifting left* helps to empower the frontline support team to detect and correct the issue or perform the task, or even better, empower the customer to self-serve and perform the task themselves. The following diagram shows how support cases start with a customer, go to a frontline support team, and then to the engineering team. An arrow indicates that we shift the resolution action to the left to reduce the impact of incidents.



Keep things standard: It can be tempting to mitigate an issue by making special arrangements for one customer. At scale, everything that's special becomes a corner case that causes something else to fail. Aim to keep all tenants using standard code, settings, and configuration.

Continuous innovation

Throughout this article, we've talked about the need to get your product to the cloud and start the virtuous cycle of continuous learning and continuous improvement.

Continuous innovation is an expectation for most SaaS products. But when a SaaS product is the successor to a long-standing on-premises product, it likely takes significant change management to prepare customers for continuous innovation.

Here are three key focus areas from our Dynamics 365 transformation:

Near-zero downtime maintenance: As the number of customers in various businesses and locations increases, it becomes impossible to find universally acceptable maintenance windows. You need to build engineering maturity so that maintenance activities can happen while the system is online. In particular, deployment of service updates needs to happen with downtime that's as close to zero as possible.

Eliminate regressions: It takes customer trust to depend on a mission-critical service with a continuous innovation policy. That trust is earned in small drops with every day of successful operation and every seamless service update. Unfortunately, it's lost in buckets—quickly and in large amounts—with any regression, no matter how small. It's worthwhile to do everything you can do that eliminates regressions in the engineering process, especially by using safe deployment processes.

Feature flags: The Dynamics 365 team has invested extensively in a [feature flags](#) framework. A feature flag can be enabled or disabled for the entire service, for subsets of tenants, or even for a single tenant. By using feature flags, we enable the introduction of new capabilities without disrupting the operation of the mission-critical business processes that Dynamics 365 supports.

Here's how feature flags can help:

- A simple performance or security fix can be introduced with the flag enabled by default. Everyone should get the benefit of the change immediately.
- Something that changes a user's experience, a business process, or the behavior of an externally visible API is introduced with the flag turned off by default.
- Changing a feature flag is effectively changing the code that runs, so feature flag changes should be managed through safe deployment as well. For example, suppose you introduce a fix for an issue, and you turn the feature flag for the fix off by default. You can enable the flag for customers that reported the original bug. You can then slowly progress through enabling the flag on widening rings of customers until it's turned on for everyone.
- If a fix is introduced when the flag is turned on by default, and the fix has a problem, it can logically be rolled back instantly by switching the flag off.
- Feature flags can also be used to selectively disclose features in preview or to selectively hide features from new customers as part of a deprecation process.
- You can provide visibility of new feature flags and tenant-specific feature flag settings to frontline support and live-site teams. This information helps teams

quickly rule in or rule out a new feature change when they investigate an issue. If necessary, teams can also adjust the settings of feature flags to mitigate the issue.

- Finally, you need to manage the lifecycle of feature flags to prevent making the code base unsupportable. Establish a process to remove feature flags from the code after they're fully deployed and proven.

Conclusion

Transitioning from an on-premises product to SaaS requires significant changes in every part of how you deliver software to customers, and in every part of your business. The cultural changes are as significant as the technical changes: moving from an occasional on-premises release cadence to a regular release cadence is a major shift, and adopting a live-site culture and processes takes effort. You need to ensure that your team is well-prepared for the journey and that you expand your talent pool beyond engineers to ensure that you have people who know how to operate SaaS at scale.

Many organizations don't survive transitions of this magnitude, especially if a new competitor comes along who is natively in the cloud already. To set yourself up for success, you should carefully scope an MVP, get it to production as quickly as possible, and then iterate and improve on it rapidly. The transition process is often the most difficult time, so it's good to make the transition to the cloud as quickly as possible.

Here are key considerations that we hope you take away from our experiences and some of the decisions you need to make for your own journey.

- **Decide on your own path.** If you have an existing application with rich functionality and an established on-premises customer base, it can make sense to move to a cloud-based SaaS model. Every product's journey is different, and you need to consider the decisions and questions separately, based on your own needs. Take inspiration from other journeys, but forge your own path.
- **Determine a strategy.** Having a product with established customers means you need to decide how to create your priorities. You might focus on building a product that works well for new customers immediately. You might focus on getting your existing customer base migrated to your new service as quickly as possible. The reason you're moving, and your capacity to make significant changes, influences the direction you take.
- **Decide whether you're going to use a single-cloud or multicloud.** Consider whether it makes more sense for you to build on top of a single cloud or to spread your engineering efforts across multiple clouds. If you're building a platform component, a multicloud strategy can make sense. If you're building an

application, a single-cloud strategy can provide benefits over using a consistent and integrated platform.

- **Plan specifically for the cloud.** A lift-and-shift approach to migrate to the cloud isn't sufficient. You need to plan how to take advantage of the elasticity of the cloud, and how to operate a service in a cloud environment. Automation, resiliency, scale, security, performance, and observability are all important considerations. You don't need to do everything upfront, but you need to know the destination so that you can plan a roadmap.
- **Plan specifically for SaaS.** A SaaS business model looks different compared to an on-premises software delivery approach. Customers expect trial accounts, understandable billing models, a fully managed service, and dynamic scale based on their needs.
- **Land, learn, and iterate.** Plan an MVP scope, identify the baseline wins that you can achieve, and then get there fast. Once you're there, commit to continuous improvement.
- **When you're in the cloud, take advantage of it.** The cloud provides many capabilities that on-premises solutions don't have. The capabilities include massive scale, elasticity, and the ability to use cloud platform components to rapidly create and iterate on your ideas. Consider how you can use technologies like generative AI, microservices, and other approaches that are hard to use outside a cloud environment.
- **Become the customer's IT department.** Customer expectations shift when you're providing an end-to-end service. Plan how you can shift left. Ensure you have a comprehensive monitoring and self-healing capability.
- **Learn from customer usage.** When you run a service, you collect a large quantity of useful data about how customers use your product. Adopt a data culture. Learn about your customers, what they do, and how they do it. Experiment and be agile enough to change your approach when the data indicates something isn't as you expect.
- **Provide ongoing value through updates.** Think about when and how to deploy updates. Plan to fix issues quickly or fall back to previous versions. Decide how to handle breaking changes. Avoid one-off changes for specific customers, because every point of difference is an opportunity for something to go wrong.

The biggest lesson we've learned is that the journey to SaaS never ends. A product roadmap is a living document that's constantly evolving. You always have many items in the backlog to improve your product, both for adding new features and for improving how you operate and deliver the product. Delivering SaaS requires continuous improvement to your processes, investment in quality, and vigilance to ensure that you provide a reliable, secure, performant, and trustworthy service for your customers.

Advances in technologies, like generative AI, bring along tremendous opportunity for you to provide capabilities that were unimaginable before.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

- [Mike Ehrenberg](#) | CTO, Microsoft Dynamics
- [John Downs](#) | Principal Program Manager
- [Arsen Vladimirskey](#) | Principal Customer Engineer

Starter web app for SaaS development

Azure App Service

Microsoft Entra External ID

Azure SQL Database

Azure Logic Apps

Azure Resource Manager

Software as a Service (SaaS) is a complex topic with many points to consider.

Independent software vendors (ISVs) who build their SaaS solutions on Azure need to solve problems and make decisions such as:

- Which [tenancy model](#) should I use?
- How do I set up an identity solution for use in a multitenant architecture?
- How do I handle onboarding new customers?

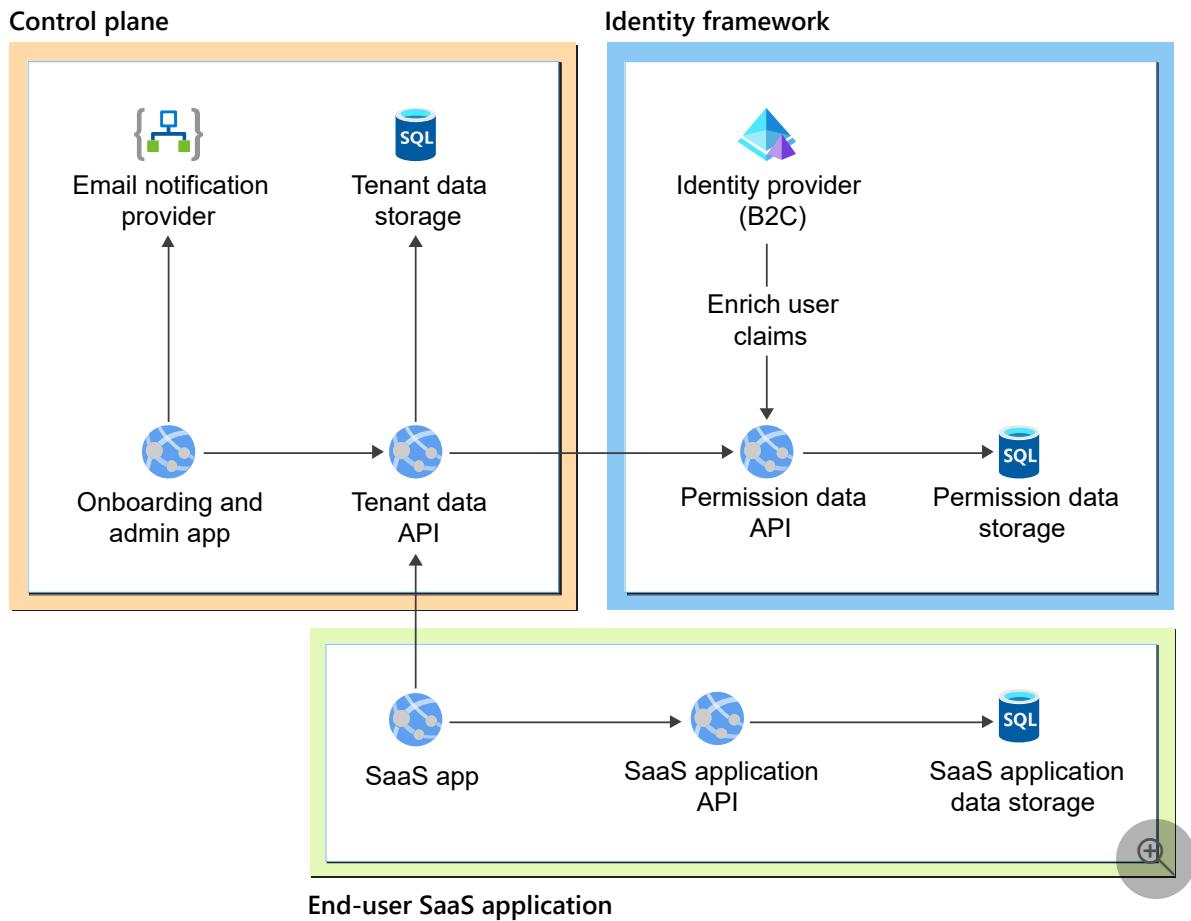
This architecture aims to answer some of these questions and provide a starting place into the world of SaaS. This architecture is adaptable to fit a wide range of scenarios.

Potential use cases

The following are some example use cases in which you could use this architecture:

- Modernize an existing application to support full multitenancy as part of a shift to a SaaS-based business model.
- Develop a completely new SaaS offering.
- Migrate a SaaS offering from another cloud service to Azure.

Architecture



[Download a PowerPoint file](#) of this architecture.

Terminology

The following table describes terms that appear in this article.

[\[+\]](#) [Expand table](#)

Term	Description	Example
SaaS vendor or ISV	The entity that owns the SaaS application and code and sells the SaaS product.	Contoso Inc, selling their SaaS application: Contoso Tickets.
Tenant	A purchased instance of the SaaS application from SaaS Vendor.	Fourth Coffee Shop.
SaaS customer admin	People who purchase or administer an application tenant.	Joe, owner of Fourth Coffee Shop.

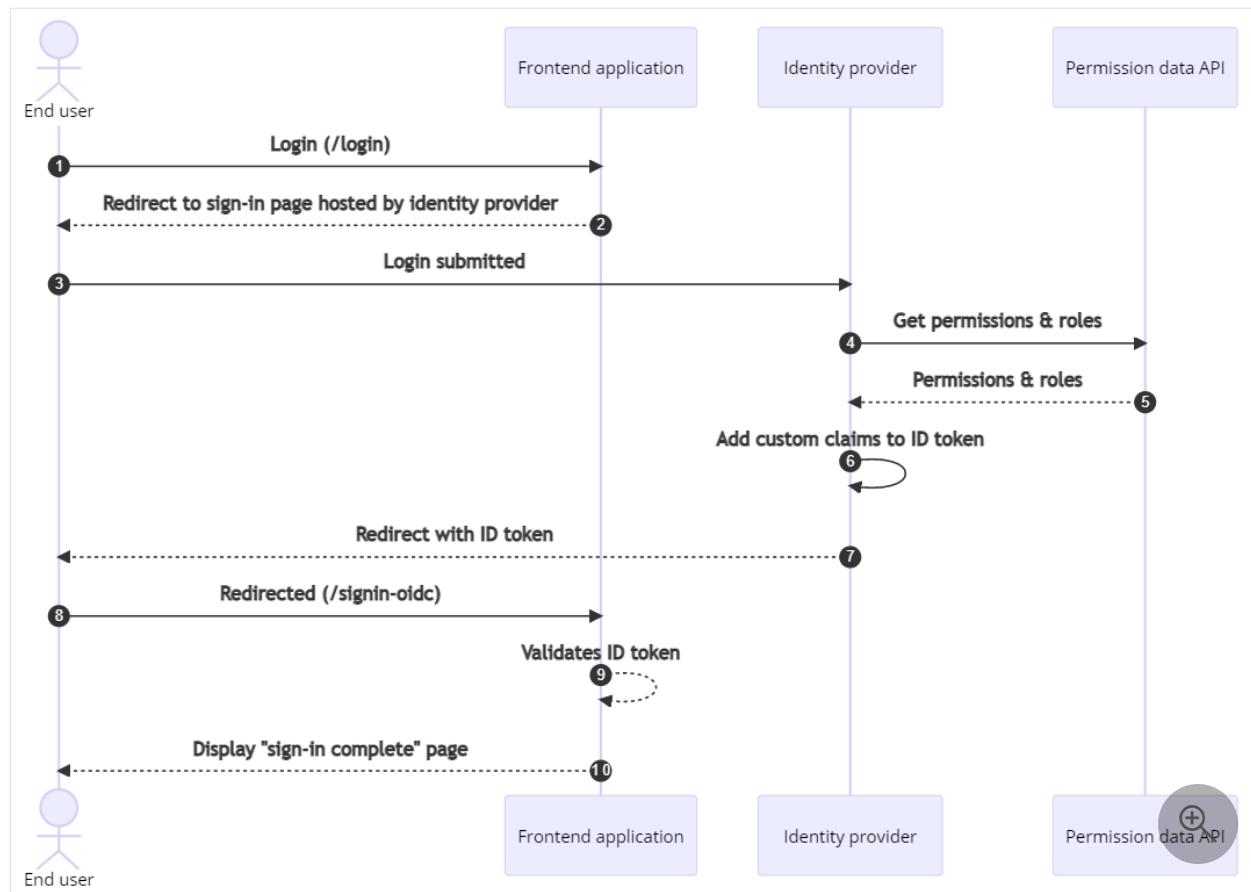
Term	Description	Example
SaaS customer user	People who use an application tenant without administering it and usually belong to the same company or group as the SaaS customer admin.	Jill, event manager at Fourth Coffee Shop, and Susan, customer of Fourth Coffee Shop.
End user	A SaaS customer admin, SaaS customer user, or any other user types that are introduced. This is a generic term to describe users who sign into the application.	Joe, Jill, and Susan are all end users (from the ISV perspective).
Front-end application	Any front-end application.	The Onboarding & admin app and SaaS app are both front-end applications.

Workflow

1. The *SaaS customer admin* navigates to the site that is hosted on the *Onboarding & admin app*.
2. The *SaaS customer admin* signs in by using the [user sign-in](#) workflow.
3. The *SaaS customer admin* completes the [onboarding flow](#).
4. The *SaaS customer admin* navigates to the tenant admin area on the *Onboarding & admin app* and [adds a SaaS Customer User](#) to their newly created tenant.
5. The *SaaS customer user* navigates to the *SaaS application app* and uses the SaaS application.

User sign-in

The user sign-in workflow consists of the following steps:



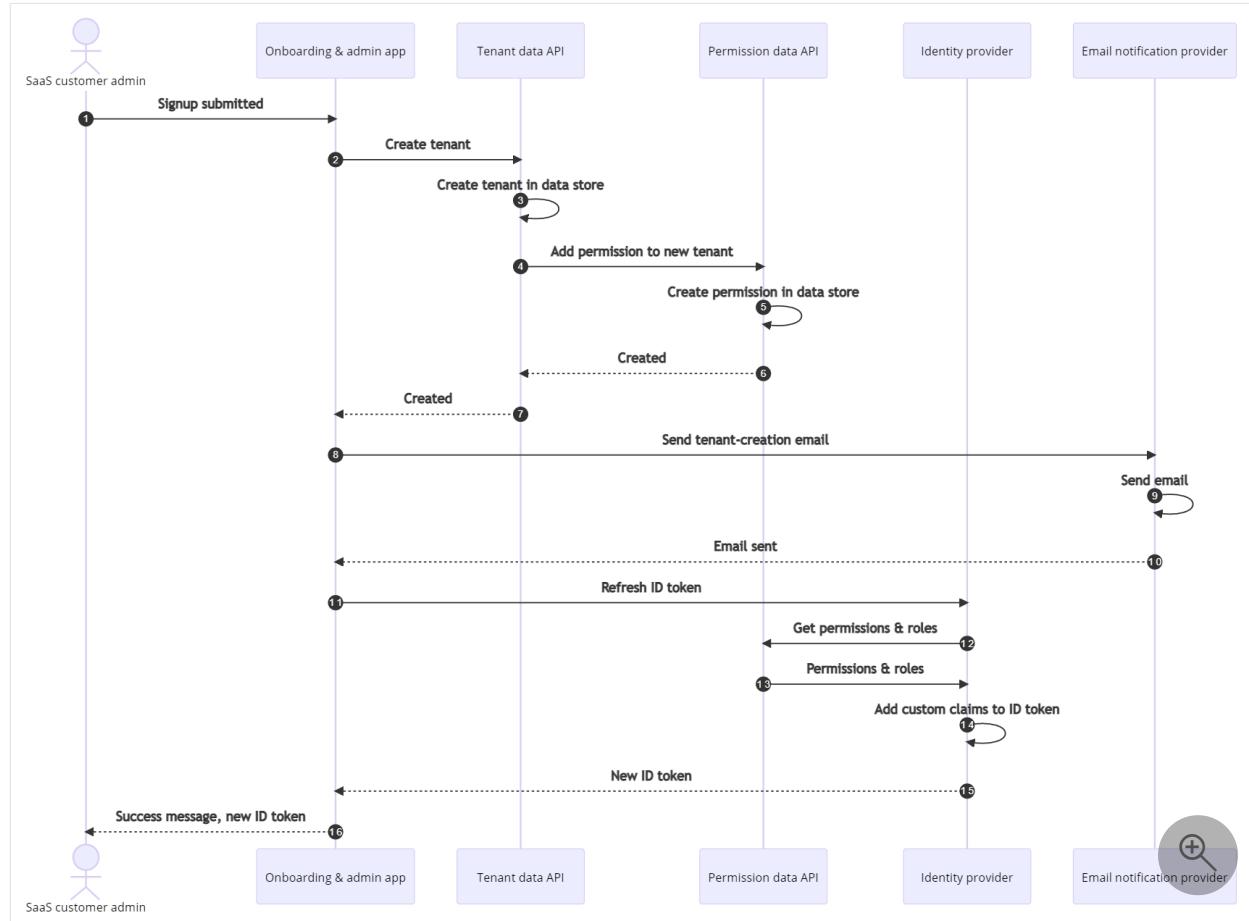
1. The *End user* navigates to a *front-end application* and selects a **Login** button.
2. The *Front-end application* redirects the *end user* to a sign-in page that is hosted by the *identity provider*.
3. The *End User* enters account information and submits the sign-in form to the *Identity provider*.
4. The *Identity provider* issues a **POST request** with the *end user*'s email address and object ID to retrieve their permissions and roles.
5. The *Permission data API* looks up the *end user*'s information in the *Permission data storage* and returns a list of permissions and roles that are assigned to that *end user*.
6. The *Identity provider* adds the permissions and roles as custom claims to the **ID token**, which is a JSON web token (JWT).
7. The *Identity provider* returns an ID token to the *end user* and initiates a redirect to the *front-end application*.
8. The *End user* is redirected to the sign-in endpoint on the *front-end application* and presents the ID token.
9. The *Front-end application* validates the presented ID token.

- The *Front-end application* returns a successful sign-in page and the *end user* is now signed in.

For more information about how this sign-in flow works, see [OpenID Connect protocol](#).

Onboard a new tenant

The tenant onboarding workflow consists of the following steps:

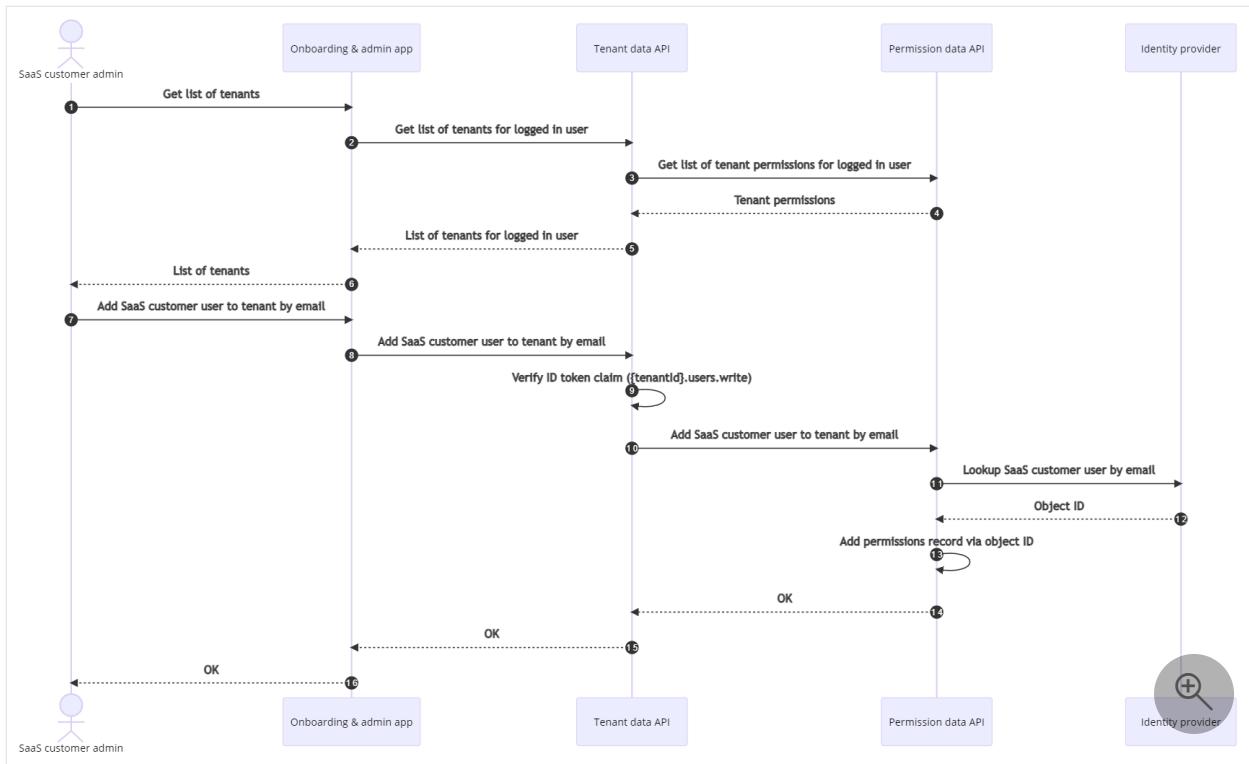


- The *SaaS customer admin* navigates to the *Onboarding & admin app* and completes a sign-up form.
- The *Onboarding & admin app* issues a POST request to the *Tenant data API* to create a new tenant.
- The *Tenant data API* creates a new tenant in the tenant data storage.
- The *Tenant data API* issues a POST request to the *Permission data API* to grant the *SaaS customer admin* permissions to the newly created tenant.
- The *Permission data API* creates a new permission record in the *Permission data storage*.
- The *Permission data API* returns successfully.

7. The *Tenant data API* returns successfully.
8. The *Onboarding & admin app* issues a POST request to the *Email notification provider* to send a "tenant created" email message to the *SaaS customer admin*.
9. The *Email notification provider* sends the email.
10. The *Email notification provider* returns successfully.
11. The *Onboarding & admin app* issues a request to the *Identity provider* to refresh the *SaaS customer admin*'s ID token so that it will include a JWT claim to the newly created tenant.
12. The *Identity provider* [issues a POST request](#) with the *SaaS customer admin*'s email address and object ID to retrieve their permissions and roles.
13. The *Permission data API* looks up the *SaaS customer admin*'s information in the *Permission data storage* and returns a list of permissions and roles assigned to the *SaaS customer admin*.
14. The *Identity provider* adds the permissions and roles as custom claims to the ID token.
15. The *Identity provider* returns the ID token to the *Onboarding & Admin App*.
16. The *Onboarding & admin app* returns a success message and a new ID token to the *SaaS Customer Admin*.

Add a user to a tenant

The addition of a user to a tenant workflow consists of the following steps:



1. The *SaaS customer admin* requests to see a list of tenants from the tenant admin area on the *Onboarding & admin app*.
2. The *Onboarding & admin app* issues a GET request to the *Tenant data API* to get a list of tenants for the *SaaS customer admin*.
3. The *Tenant data API* issues a GET request to the *Permission data API* to get a list of tenants that the *SaaS customer admin* has access to view.
4. The *Permission data API* returns a list of tenant permissions.
5. The *Tenant data API* looks up the tenant information in the Tenant data storage and returns a list of tenant data based on the list of tenant permissions received.
6. The *Onboarding & admin app* returns the list of tenant data to *SaaS customer admin*.
7. The *SaaS customer admin* selects a tenant from the list to add a *SaaS customer user* to and enters the email address for the *SaaS customer user*.
8. The *Onboarding & admin app* issues a POST request to the *Tenant data API* to add a permission for the *SaaS customer user* on the specified tenant.
9. The *Tenant data API* verifies that the *SaaS customer admin* has a valid JWT claim to the specified tenant and has the user's write permission on it.
10. The *Tenant data API* issues a POST request to the *Permission data API* to add a permission for the *SaaS customer user* on the specified tenant.

11. The *Permission data API* issues a GET request to the *Identity provider* to look up the *SaaS customer user* by the provided email address.
12. The *Identity provider* returns the *SaaS customer user's* object ID.
13. The *Permission data API* adds a permission record in the *Permission data storage* for the *SaaS customer user* on the specified tenant by using their object ID.
14. The *Permission data API* returns successfully.
15. The *Tenant data API* returns successfully.
16. The *Onboarding & admin app* returns successfully.

Components

This architecture uses the following Azure services:

- [App Service](#) ↗ enables you to build and host web apps and API apps in the programming language that you choose without needing to manage infrastructure.
- [Azure Active Directory B2C](#) ↗ easily enables identity and access management for end user applications.
- [Azure SQL Database](#) ↗ is a general-purpose relational database managed service that supports relational data, spatial data, JSON, and XML.
- [Azure Logic Apps](#) ↗ lets you quickly build powerful integrations using a simple GUI tool.

Alternatives

The effectiveness of any alternative choices depends greatly on the [tenancy model](#) that you intend for your SaaS application to support. The following are some examples of alternative approaches that you can follow when you implement this solution:

- The current solution uses Azure Active Directory B2C as the identity provider. You could instead use other identity providers, such as [Microsoft Entra ID](#) ↗ .
- For stricter security and compliance requirements, you could choose to implement private networking for cross-service communication.
- Instead of using REST calls between services, you could implement an [event-driven architectural style](#) for cross-service messaging.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can follow to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Overview of the security pillar](#).

This solution relies on identity as its security paradigm. Authentication and authorization for the web apps and APIs is governed by the [Microsoft identity platform](#), which is responsible for issuing and verifying user ID tokens (JWTs).

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

The components in this solution have some cost associated with their operation, but the cost is modest for most web applications and SaaS solutions. Also, you can control the cost by managing the following resource settings:

- You can scale the App Service plan that runs the application to fit the throughput that you need. In addition, you could run each app on a separate plan if you require higher throughput, but you'll incur a higher cost as a result. For more information, see [Azure App Service plan overview](#).
- Azure AD B2C provides two SKUs: Premium P1 and Premium P2. Both SKUs include a free allowance for the number of monthly active users (MAUs), but you need to evaluate which features that each SKU provides to determine which is required for your use case. For more information, see [Microsoft Entra External ID pricing](#).
- Azure SQL has several purchasing models to fit a wide array of use cases, including the ability to autoscale. You need to evaluate the usage on your own databases to ensure you size them correctly. For more information, see [Compare vCore and DTU-based purchasing models of Azure SQL Database](#).

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. For more information, see [Overview of the performance efficiency pillar](#).

This architecture should be able to scale to easily meet most medium to medium-large workloads. Since the architecture mostly uses Azure's platform (PaaS) services, you have many options to adjust the scale of the solution based on your requirements and load.

Deploy this scenario

If you'd like to deploy this scenario, see the [Azure SaaS Dev Kit](#) on GitHub. It's a deployable reference implementation of this architecture.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Landon Pierce](#) | Customer Engineer

Other contributors:

- [Chris Ayers](#) | Senior Customer Engineer
- [John Downs](#) | Senior Customer Engineer
- [LaBrina Loving](#) | Principal SVC Engineering Manager
- [Gary Moore](#) | Programmer/Writer
- [Nick Pinheiro](#) | Senior Consultant
- [William Salazar](#) | Senior Customer Engineer
- [Ali Sanjabi](#) | Senior Customer Engineer
- [Arsen Vladimirsksiy](#) | Principal Customer Engineer
- [Jason Young](#) | Principal SVC Engineering Manager

Next steps

Here are some additional recommended resources for building a SaaS application on Azure:

- [Architect multitenant solutions on Azure](#) - Describes best practices.
- [ISV Considerations for Azure landing zones](#)
- [Microsoft Azure Well-Architected Framework](#)

- [WingTips Tickets SaaS application](#) - Provides details about tradeoffs between various tenancy models within the database layer.

Related resources

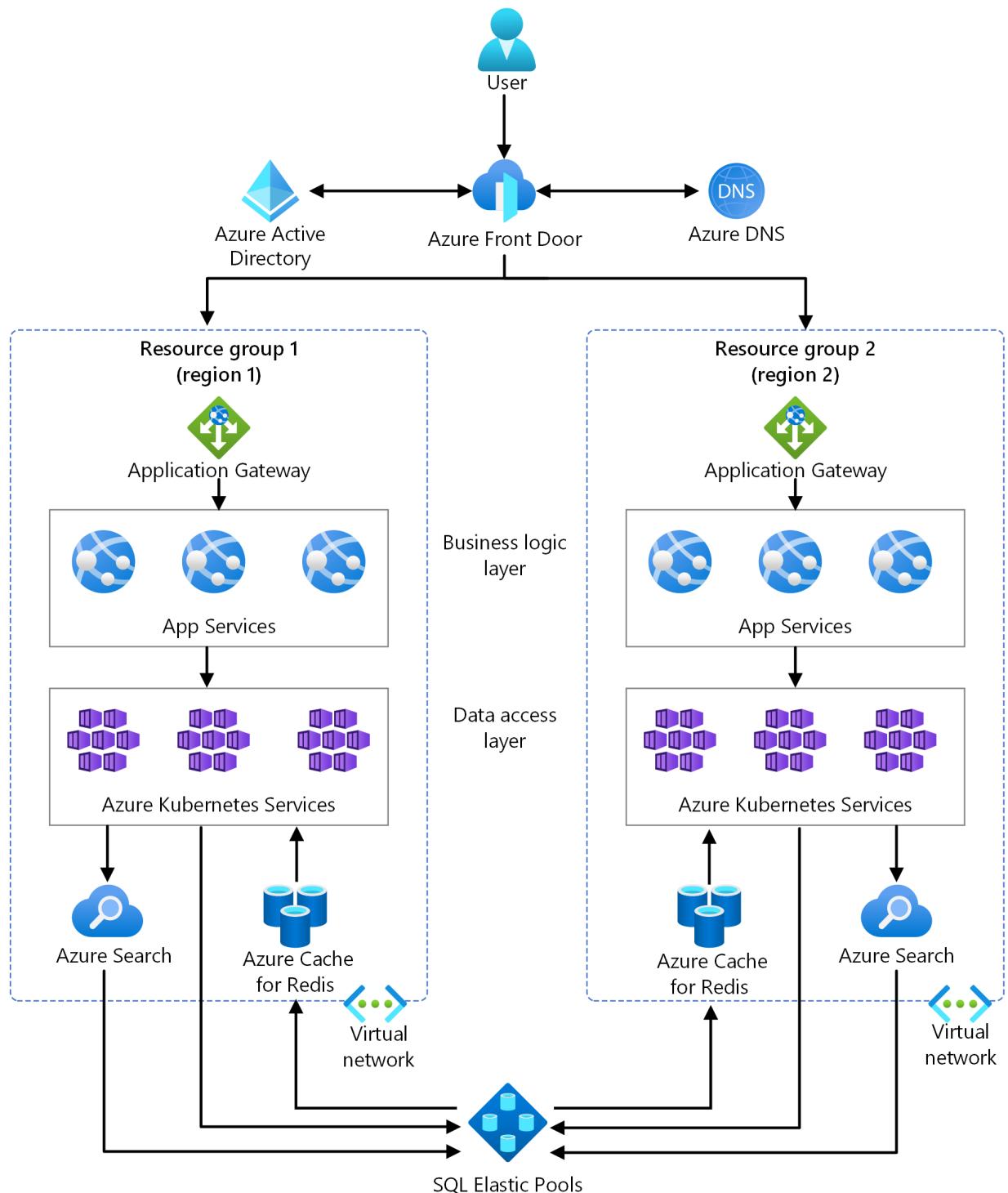
- [Tenancy models to consider for a multitenant solution](#)
- [Architectural approaches for compute in multitenant solutions](#)
- [Architectural approaches for storage and data in multitenant solutions](#)
- [Azure App Service and Azure Functions considerations for multitenancy](#)
- [Multitenant SaaS on Azure](#)
- [Cloud Design Patterns](#)

Multitenant SaaS on Azure

Microsoft Entra ID Azure App Service Azure DNS Azure Front Door Azure Kubernetes Service (AKS)

When you identify a portion of your business's software solution that you can unbrand and market to other businesses, it adds an entire new revenue stream for a company. However, configuring the solution to account for the load that a slew of tenants brings is often a challenging obstacle to tackle. This solution tours a suite of Azure technologies that secure and balance the traffic.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

A suite of Azure technologies secure and load balance the traffic.

1. Microsoft Azure Front Door handles a few initial tasks:

- Processing the initial request

- Load balancing across the regions
- SSL(HTTPS) termination and offloading
- Failing over if there's a regional outage

2. Azure DNS manages DNS records and ensures routing to the correct Azure Front Door endpoint.
3. The architecture uses Microsoft Entra ID as the identity provider for authentication.
4. Once routed to the appropriate region, Application Gateway routes and load balances, directing requests to the appropriate Azure App Service.
5. For this architecture, using App Service is the preferred service for:

- Any HTTP-based application.
- Serving web content.
- Exposing RESTful APIs.
- Implementing business logic behind the front-end application.

You can configure App Service to scale up and out automatically. It makes App Service a good fit for scaling a host of tenant HTTP-driven requests on demand.

6. The data-access layer services are also independently scaled based on load. Data services manage data models, connection clients, and drivers. The services also provide a consistent data interface for all higher-level services wishing to consume data in the application. You can deploy and scale these data services using the Azure Kubernetes Service (AKS). Each AKS cluster is responsible for a set of related features in the layer. AKS can implement a microservice architecture, which features a series of containers that each encapsulate specific functionality within the cluster. This allows for a high degree of abstraction and de-coupling within the code. It also allows for clusters to scale out individually to account for increased load from multiple tenants. Each cluster can scale up its resources if load increases on the cluster. The scale up doesn't affect the other clusters in the resource group as long as they aren't experiencing the same increase.
7. Store and manage relational data outside of the application framework. Doing so provides a single point of data entry for either region. You can achieve replication, availability, scalability, and security by leveraging the strength of Azure SQL Elastic Pools. Provision each tenant a database in a pool. Allocate the resources available

in the pool to databases on-demand as load and requests come in. This optimizes database resources available for tenants against your budget.

Components

The primary components are the suggested components for the architecture in this solution. If any of the primary components don't fit your architecture, see the list of alternative components.

Primary components

- [Azure Front Door](#): A regional load balancer that routes client traffic to the correct region. It can fail over to the second region if a regional failure happens, and it can secure the internet-facing entry point via [Azure Web Application Firewall](#).
- [Microsoft Entra ID](#): Acts as the identity provider for the entire application, enforcing authentication and end-to-end authorization of the request in the application.
- [Azure DNS](#): A hosting service in Azure for domain name resolution. In a multitenant solution, multiple clients access the solution via their own individual domains. Use Azure DNS to configure and resolve client requests to their correct application stack.
- [Application Gateway](#): Routes and load-balances traffic internally in the application to the various services that satisfy client business needs. While Azure Front Door balances load across high-level regions, it's Application Gateway that has awareness of the load on individual services within a group. Azure Front Door and Application Gateway combine to provide complex load-balancing at all levels in a multitenant solution. For more information on load-balancing options in Azure, visit this [overview on Azure load-balancing](#).
- [App Service](#): Azure's premier service for web applications and web-based APIs. Security integrates with services like Microsoft Entra ID and [Azure Key Vault](#). You can configure automatic scaling. Also, the amount of resources available to scale to is flexible between the various App Service plans on which the app can run. App Service can also leverage integrated DevOps capabilities for continuous integration and deployment to multiple environments. These and other supporting features of the Azure platform allow for developers to focus on the development of their applications.

- [Azure Kubernetes Service \(AKS\)](#): Orchestrates instances of container images deployed to a cluster. Managing multiple clients' data often involves implementing a suite of components to manage:
 - Data modeling
 - Data source connectivity
 - Extract, transform, load (ETL)
 - Import/export activities

Developing these many smaller components as container-based microservices creates an ideal scenario for the deployment to an AKS cluster. Tools for autoscaling, load balancing, and upgradeability are built into the framework. AKS integrates well with a continuous integration/continuous delivery (CI/CD) strategy using the available DevOps features and Azure Container Registry.

- [Azure SQL Elastic Pools](#): Provides a solution for managing a set of databases flexibly with a pool of resources. The service allocates resources on demand to the databases. It gives the developer of a multitenant SaaS architecture the power to deliver database resources to clients as they need it. The service also reduces the budget and overhead of maintaining multiple SQL Servers with large chunks of unused compute resources.
- [Azure Cognitive Search](#) (formerly known as Azure Search): A service that adds a powerful indexing and query engine to your application. It gives clients access to strong query functionality. They can also use Azure's AI capabilities to enrich and enhance the query functionality. Azure Cognitive Search can account for multitenancy using an index-per-tenant or service-per-tenant strategy.
- [Azure Cache for Redis](#): Applies a caching layer as a service to the solution, providing an in-memory managed cache to reduce latency and increase performance for the clients. High throughput allows for a high volume of requests to handle multiple tenants accessing the system. You can flexibly scale up the service as application loads increase. It also supports encryption at rest to protect and isolate cached tenant data.

Alternatives components

- [Azure Virtual Machine Scale Sets](#): Allows for the deployment of services to a virtual machine (VM) environment that scales and grows automatically as needed. Virtual Machine Scale Sets integrate well with a [Load Balancer](#) or Application

Gateway to automatically rebalance load as the scale set grows. Virtual Machine Scale Sets provides the scalability this solution demands. In many cases though, it's unnecessary to manage the full VM environment, and we can defer that level of the stack to App Service or AKS.

- [Azure SQL Database](#): Implement as individual dedicated instances as a replacement for Elastic Pools. Using Azure SQL Database adds higher overhead in managing the instance directly and incurs more cost for allocated resources. That said, it's an acceptable alternative when the tenant requires a dedicated server. In particular, the client might require more control over the instance and dedicated available resources. Tenants that require a dedicated SQL Server can exist side by side with tenants on an Elastic Pool configuration. You can make a tier of SQL databases one of the pricing options available to tenants when purchasing licenses for the SaaS.
- [SQL Server on Virtual Machines](#): Another option for the deployment of SQL databases. The tenant might have pre-existing IT infrastructure and existing SQL Servers on premises. In that case, the tenant might want to use their current licenses, either as a full migration or in a hybrid scenario. The decoupled nature of the SaaS allows for the data layer of the application to target any SQL Database via configuration.

Scenario details

When you identify a portion of your business's software solution that you can unbrand and market to other businesses, it adds an entire new revenue stream for a company. However, configuring the solution to account for the load that a slew of tenants brings is often a challenging obstacle to tackle.

Azure offers a range of services for managing a software solution that:

- Flexibly maintains databases for all clients.
- Scales the business and logic tier of the solution to prevent bottlenecks at the compute layer.
- Integrates availability and regional failover.
- Provides end-to-end security at all levels of the solution.

Potential use cases

These use cases have design patterns that can benefit from a multitenant SaaS solution hosted on Azure:

- Develop a customer relationship management (CRM) solution that clients can market and sell to customers.
- Implement a content management system (CMS) system and deliver it to multiple users using this architecture.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Multitenancy

A multitenant solution is the key consideration in this solution. The solution handles a number of clients simultaneously. It also allocates enough resources to process all client requests effectively. While processing requests, the solution secures traffic from global endpoints and isolates client data to prevent breaches and cross-contamination. Deploy clients to a pair of regional resource groups based on their primary location. Doing so optimizes regional availability.

You can deploy many clients to a single compute group because the system isolates requests based on authentication and client keys, which differentiates requests based on these unique identifiers. The system can encrypt all client requests separately by their keys so that no client can decrypt any other client's data. Managing multiple clients on a single compute stack gives you the ability to optimize resource allocation to provide clients the responsiveness they need at cost.

You manage client databases in a similar way outside of the compute stack, because a client request could arrive from either of the regional stacks. Many client databases can exist on the same Elastic Pool, isolated and secured by transparent data encryption (TDE). You can configure each database to encrypt data using a client-managed key and decrypt the data just in time (JIT). Decrypting JIT protects client data from both the developer and other clients. The system leverages the Elastic Pool to provide resources on demand to the clients assigned to it while keeping costs low for you. You can assign replication policies to each Elastic Pool to provide backup and failover for client data. Bring more Elastic Pools online as you onboard more clients into the system.

For more information about multitenant solutions, see [Architect multitenant solutions on Azure](#).

Reliability

Reliability ensures your application can meet the commitments you make to your customers. For more information, see [Overview of the reliability pillar](#).

Scalability and Availability

This solution is designed to account for a large number of tenants using the SaaS. It takes advantage of the large number of scalable components and services to grow based on load. This architecture isn't designed for solutions that service a few tenants, or a small load of requests and data. It could stress the budget of a solution targeting a single client or smaller load. It's also unnecessary to have the multiregion overhead where high global availability isn't a requirement, because it adds unnecessary complexity and cost.

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Overview of the security pillar](#).

The system addresses security from end-to-end at each level of the application:

- Azure Front Door provides built-in HTTPS support for its domains. This means the system can encrypt all traffic to the SaaS application. Azure Front Door also implements Azure Web Application Firewall, protecting the SaaS stack from attacks at the edge, before the system routes requests to the application.
- Each application stack in each region lies within an Azure Virtual Network. The system restricts traffic into the virtual network accepting requests from Azure Front Door, protecting all application services from external traffic. Once inside the secure firewall, Application Gateway can terminate SSL and provide performant load balancing and routing within the application.
- You can securely manage all credentials, secrets, and connection strings by using Azure Key Vault. By managing this sensitive data as secrets, developers can inject credentials into the application at the time of deployment. Doing so makes sure that the code isn't polluted with sensitive information. Using secrets protects client data by ensuring that a breach in code or man-in-the-middle attack wouldn't gain access to tenant databases.

- In this scenario, the data of multiple tenants might exist side by side on the same database server, if not the same database. Using TDE and JIT decryption protects data on the database. The system encrypts all data on the database at rest, and only decrypts it when requested by the tenant. Clients can provide their own keys, and you can store all client keys in Azure Key Vault to manage encryption for multiple tenants. It protects client data end to end, prevents the developer from having access to client data, isolates data between tenants, and helps to meet compliance requirements for security and data.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

Azure App Service provides many pricing tiers based on the expected compute resources required. For a multitenant SaaS, high availability and scale-out capabilities are key components in choosing the service plan. If you expect to host many tenants, choosing a premium or isolated tier might be necessary to provide the compute resources necessary to account for the high traffic. The standard, premium, and isolated tiers are all dedicated VM instances. You can calculate cost per unit of time by how many VMs of said tier you've specified. For more information, visit the [overview of App Service pricing plans](#).

Azure Kubernetes Service provides a cost-effective container service. Charges for AKS nodes only occur on usage, so you're only charged for:

- The VMs
- Consumed storage and network resources
- Scaling cost directly related to usage

Using AKS as the data-tier service is ideal if you're looking to reduce costs. For an estimate on pricing out a layer of AKS instances, visit the [Kubernetes service calculator](#).

By design, the Azure SQL Elastic Pool pricing is highly cost-effective in a multitenant scenario. Tenant databases in an Elastic Pool will share the available resources. As demand shifts between tenants over time, resources will shift as well. Azure SQL Elastic Pool provides the maximum available resources to demanded databases without the need for resource overhead on all databases. The service keeps cost low for the developer of the SaaS and the tenants. Use the [Azure SQL Database pricing calculator](#)

to price out and determine the tier and amount of resources needed to serve your tenants and their data.

- Using a virtual core (vCore) pricing model provides greater flexibility in scaling to meet required resources. Also, you can take advantage of the Azure Hybrid Benefit. Existing SQL Server licenses provide a discount to vCore SQL resources in the cloud. Therefore, in an instance when on-premises servers are already part of the developer infrastructure, you can manage cost even more by using these discounts. You can estimate your potential savings by using the [Azure Hybrid Benefit savings calculator](#).
- You can also save cost on SQL Server resources by purchasing [Azure SQL Database reserved capacity](#). Purchasing reserved capacity marks a commitment of long-term SQL Database usage. The term is usually between one to three years. In return, you get discounts on the compute costs of the resources in reservation. For instance, you could reserve 32 general purpose vCores for a year, which reduces the cost of those 32 vCores for that year. Having multiple tenants purchasing licenses for a SaaS is a strong indicator that making use of Reserved capacity fits the solution, and an ideal cost saver in this workload.

You can find the pricing structure for Azure Cache for Redis on the [Azure Cache for Redis pricing](#) page. Adjust the cache tier at any time between a Basic, Standard, and Premium tier based on need. You'll see higher pricing on the larger cache limits and additional features such as replication and disaster recovery. Azure Cache for Redis also offers reserved capacity pricing for long-term usage commitments.

Azure Front Door's pricing depends on the amount of data transfer in and out of the service. For outbound data, the pricing is different based on zones. Different regions will incur different costs. If you come across a price differential, estimate the cost separately. The price includes some routing and domain capacity, but the system incurs costs past the initial limits. Azure Web Application Firewall incurs a small additional charge per policy or rule applied. You can find the pricing details for Azure Front Door on the [Azure Front Door pricing](#) page.

The [pricing for Azure Cognitive Search](#) is a fully tiered system. A free tier is available for development and testing. After that, each tier incurs a per-hour cost for each Cognitive Search instance allocated. As the tiers increase, the total storage, number of indexes, and scale-out limits also increase. Azure Cognitive Search provides image extraction as a service at the same rate to all paid tiers.

Next steps

- Application and service principal objects in Microsoft Entra ID is about leveraging Microsoft Entra ID to implement multitenant apps.
- Multitenant SaaS database tenancy patterns covers implementing multitenancy patterns in SQL.
- Tenancy in Microsoft Entra ID is also about leveraging Microsoft Entra ID to implement multitenant apps.

Related resources

- Run a web application in multiple Azure regions for high availability is a reference for the multiregion requirement of the solution.
- Multitier web application built for high availability and disaster recovery on Azure is a similar example workload scenario. It describes many of the considerations for a large-scale application on Azure.

Architecture for startups

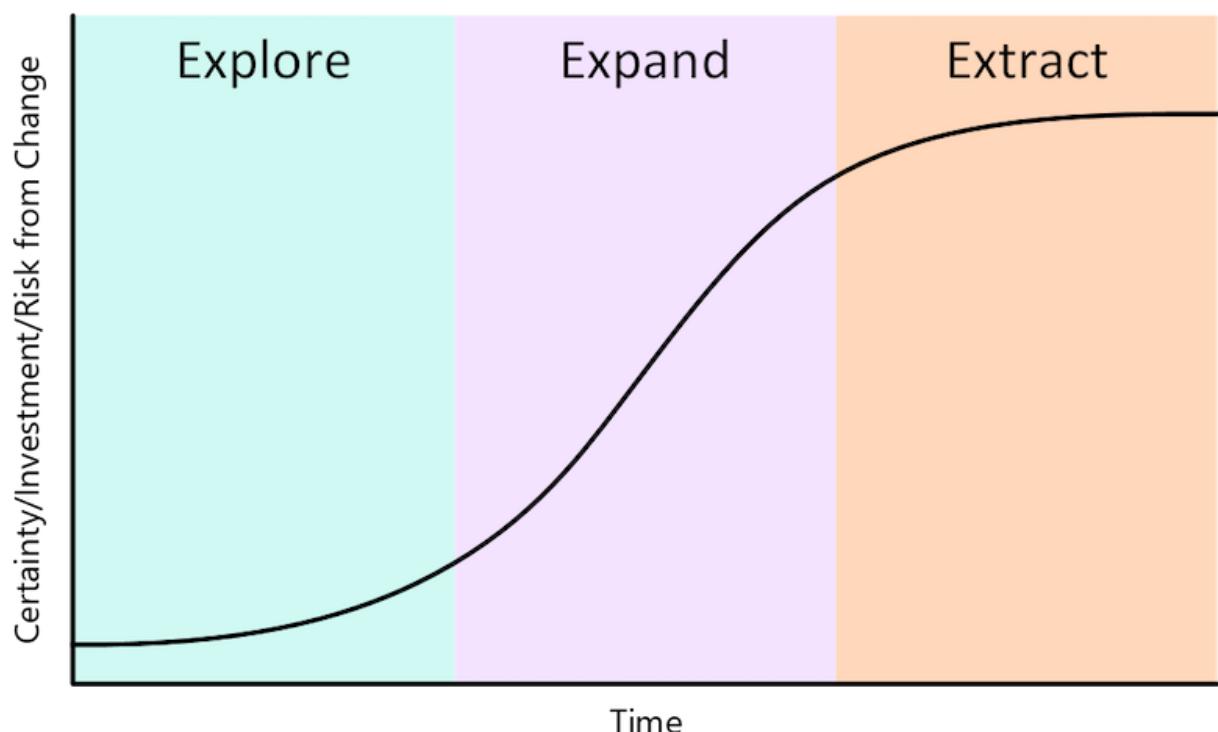
Article • 06/30/2023

Building a startup is a unique challenge. The core task is to find a place for an innovation as a product or service in the market. This process requires testing multiple assumptions that are built into the innovation. A successful startup must iterate through these assumptions, and grow and scale as its product gains product and market fit. After finding this fit, the startup must scale to meet market demands.

In the different startup life stages, developers, architects, and chief technical officers (CTOs) handle distinct phases of development. These stages require fundamentally different approaches and different technology choices. Part of the task is to establish which phase your startup is in. Choose the technologies and architectures that match that phase.

Innovation stages

Kent Beck describes a [three-stage process](#) of software product innovation. Those stages are *explore*, *expand*, and *extract*. You can think about the different parts of this process as a graph:



- The **Explore** stage starts with a low slope, where you're trying to find what works. Certainty is low, you only invest small amounts, and the risk from any changes you make is also low.

- At some point, the graph rises more rapidly. This rapid growth is the **Expand** stage. Your certainty greatly increases, you invest much more, and you're much more aware of risks.
- Finally, as the graph flattens out, you reach the **Extract** stage. The certainty, investment, and risk from change are all high, but the rate of growth has reached a plateau.

Explore

When your startup is in the exploration stage, your imperative is to invest small amounts of time and effort on many different product ideas. The fact that most ideas won't be right drives exploration. Only by iterating and learning can you find product and market fit. By making many small bets, you aim to find a product idea that pays off.

This stage requires discipline. It's easy to overinvest in an idea that you could test with less time and energy. A technologist finds it especially easy to fall into this trap. To make architectural choices that ease exploration, remember that you're exploring. You don't yet know if the current product idea is one that will scale.

From an architecture perspective, choose services that optimize for speed, cost, and options. Use managed services and platforms as a service (PaaS) like Azure App Service to get started quickly without worrying about complex infrastructure. Manage costs by choosing free tiers and smaller instance sizes while you're exploring. Containers support developing with whatever tools make sense for you and give you flexible deployment options for the future.

Build your first stack

As with your first product version, your first technology stack should be firmly rooted in exploration. That means the technology stack should ease rapid product iteration without wasting effort. You don't want to spend time or effort on infrastructure or architecture that isn't required for answering current questions.

During the exploration phase, you need to optimize for speed, cost, and optionality. Speed is about how fast you can build and move forward with an idea, or move onto the next idea. Cost is how much you're spending to run your infrastructure. Optionality describes how fast you can change directions given the current architecture.

It's important to balance cost, speed, and optionality. Too much focus on cost limits speed and optionality. Too much focus on speed can lead to increased costs and fewer

options. Designing for too many options builds complexity, which increases costs and reduces speed.

Consider using our [suggested first technology stack](#). This architecture uses PaaS services for ease of implementation, can be started with a minimal scale, and uses container and open source technologies that can easily be deployed on different technology stacks as you mature.

Expand

Once your startup finds growth through exploration, you shift gears to expansion. You focus on removing any blockages to your product's and company's continued growth. From a technical perspective, you solve infrastructure scale challenges and increase development velocity. The goals are to meet your new customers' needs and advance your product roadmap.

Extend your architecture

As you iterate on your product, you'll inevitably find areas where your architecture needs extending. You might need to complete long-running tasks in the background, or handle frequent updates from internet-of-things (IoT) devices. You might need to add full-text search or artificial intelligence to your product.

You might need architectural changes to accommodate items on your roadmap. Resist the temptation to make those changes too far in advance. Extensions risk adding complexity to your architecture and infrastructure costs to your balance sheet.

In early startup stages, any architecture extension should be just-in-time. The extension should take only as much time and energy as needed to test the next hypothesis. Be ready to remove extensions to reduce complexity. Look for product features that your customers aren't using as opportunities to simplify your architecture and reduce your infrastructure spending.

Your architecture could be expanded in many ways, such as:

- Enhancing resiliency through a [zone-redundant deployment](#)
- Enhancing resiliency through a [highly available multi-region deployment](#)
- Enhancing security through a [network hardened technology stack](#)

Extract

In the extraction phase, the pace of growth slows as you reach the limits of the market opportunity. Because you expanded through the previous phase, there's now a lot to lose, so you take a more cautious approach. Margin expansion, cost reduction, and efficiency improvements characterize the extraction phase. During the extraction phase, be careful not to compromise the product for the customers you won in the expansion phase.

Handle growth and mature your stack

Once a product achieves product and market fit, many demands drive its architecture. Increased usage might require infrastructure scaling to handle load. New enterprise compliance requirements might require greater isolation. These changes are common steps in maturing a successful application.

Changes you make to handle growth and add maturity are different from extending architecture. These changes aren't functional requirements, but relate to unlocking scale. Increased scale can come from net new customers, increased usage from existing customers, and customers with higher regulatory requirements.

Resist the temptation to optimize prematurely. Make sure to take growth and maturation steps that can help you continue iterating and improving your product.

Next steps

- See and deploy an example [Core startup stack architecture](#).

Related resources

- [Best practices in cloud applications](#)
- [Ten design principles for Azure applications](#)

Core startup stack architecture

Azure App Service Azure Blob Storage Azure Content Delivery Network Azure Database for PostgreSQL
Azure Virtual Network

Many lessons you learn in larger companies aren't directly applicable to a startup's first stack. In a product's initial [explore](#) stage, you need to optimize deployment for speed, cost, and *optionality*. Optionality refers to how fast you can change directions within a given architecture.

A business in the [expand](#) or [extract](#) phases of product development might use a service-oriented or microservices architecture. This type of deployment architecture is rarely right for a startup that hasn't yet found product/market fit or commercial traction.

For a core startup stack, a simple monolithic design is best. This design limits the time spent managing infrastructure, while providing ample ability to scale as the startup wins more customers.

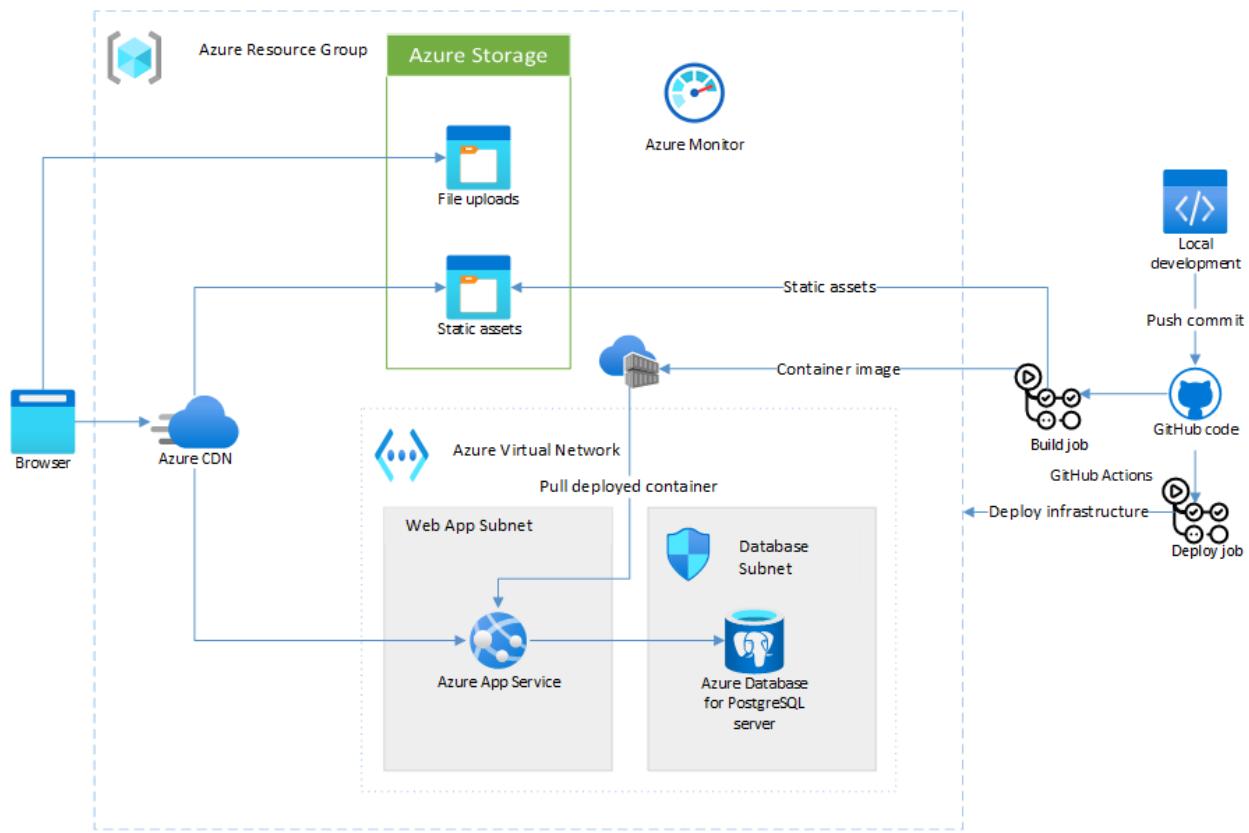
Potential use cases

This article presents an example of a simple core startup stack, and discusses its components.

Architecture

A startup, Contoso, has built an application prototype with a [Ruby on Rails](#) back end and a [React](#) front end written in [TypeScript](#). The Contoso team has been running demos on their laptops. Now they want to deploy their app for their first customer sales meetings.

While the app is ambitious, it doesn't yet need a complex, microservice-driven architecture. Contoso opted for a simple monolithic design that includes the recommended startup stack components.



[Download a Visio file](#) of this architecture.

Dataflow

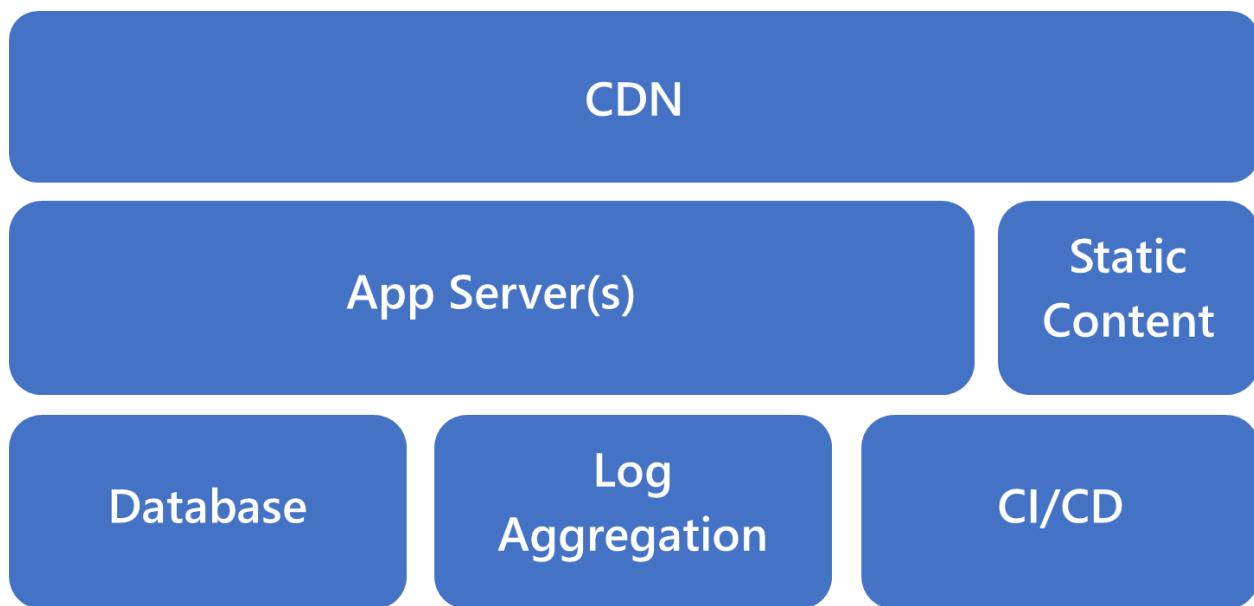
In this core startup stack architecture:

- [Azure App Service](#) provides a simple app server to deploy scalable applications without configuring servers, load balancers, or other infrastructure.
- [Azure Database for PostgreSQL](#) is an Azure database service for a leading open-source relational database management system (RDBMS). You can concentrate on developing your application rather than managing database servers.
- [Azure Virtual Network](#) segments network traffic and keeps internal services protected from internet threats. Your app servers use [virtual network integration](#) to communicate with the database without exposure to the internet.
- [GitHub Actions](#) builds continuous integration and continuous deployment (CI/CD) into your source code management. GitHub Actions has extensive support for different languages, and strong integration with Azure services.
- [Azure Blob Storage](#) stores static assets and moves load away from the app servers.
- [Azure Content Delivery Network \(CDN\)](#) accelerates content delivery to users through a global network.
- [Azure Monitor](#) monitors and analyzes what's happening across your application's infrastructure.

Core startup stack components

A complex stack can generate bugs that require constant attention. A sophisticated architecture might detract from building your product. Bugs aren't caused by complexity, but a complex stack makes it easier to ship bugs. Not all sophisticated architectures are a waste of energy, but they waste your resources if you haven't yet found product/market fit. Your first startup stack should be simple and get out of your way, so you can concentrate on product development.

The following simple diagram shows the recommended core startup stack. These components are enough to get your product off the ground and into the hands of your customers. For 80 percent of startups, this stack is all you need to test the basic hypotheses built into your product. Startups working in machine learning, internet of things (IoT), or highly regulated environments might require more components.



CDN

With few customers at the start, a CDN might seem premature. However, adding a CDN to a product already in production can have unexpected side effects. It's best to implement a CDN up front. A CDN caches static content closer to your customers, and provides a façade behind which you can iterate on your APIs and your architecture.

App server

Your code needs to run somewhere. Ideally, this platform should make deployments easy, while requiring the least possible operational input. The app server should scale horizontally, but some manual scaling intervention is fine while you're still in the explore stage.

Like most of this stack, the app server should essentially run itself. Traditionally, the app server was a virtual machine, or a web server instance running on a bare-metal server. Now, you can look to platform-as-a-service (PaaS) options and containers to remove operational overhead.

Static content

Serving static content from your app server wastes resources. Once you configure a CI/CD pipeline, the work to build and deploy static assets with each release is trivial. Most production web frameworks deploy static assets with CI/CD, so it's worthwhile to start out by aligning with this best practice.

Database

Once your app is running, you need to store your data in a database. For most cases, a relational database is the best solution. A relational database has multiple access methods and the speed of a time-tested solution. Relational databases include [Azure SQL Database](#), [Azure Database for PostgreSQL](#), and [Azure Database for MariaDB](#). Some use cases need a document database or NoSQL database like [MongoDB](#) or [Azure Cosmos DB](#).

Log aggregation

If something goes wrong with your app, you want to spend as little time as possible diagnosing the problem. By aggregating logs and using application tracing from the beginning, you help your team focus on the problems themselves. You don't have to access a file on the app server and pore over logs to get monitoring data.

CI/CD

The lack of repeatable and rapid deployments is one of the worst impediments to speed when you're iterating on a product. A well-configured CI/CD pipeline streamlines the code deployment process on your app server. Quick and easy deployments mean that you see the results of your labor quickly. Frequent integration avoids divergent code bases that lead to merge conflicts.

Deploy this scenario

The concepts and techniques are the same for most projects you build by using a [Dockerfile](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Andrew Harvey](#) | CTO and Startup Advocate

Next steps

- [Automate your workflow with GitHub Actions](#)

Related resources

- [Architecture for startups](#)
- [Best practices in cloud applications](#)
- [Best practices for using content delivery networks \(CDNs\)](#)
- [Ten design principles for Azure applications](#)

Architect multitenant solutions on Azure

Article • 10/10/2023

A multitenant solution is one used by multiple customers, or *tenants*. Tenants are distinct from users. Multiple users from a single organization, company, or group form a single tenant. Examples of multitenant applications include:

- Business-to-business (B2B) solutions, such as accounting software, work tracking, and other software as a service (SaaS) products.
- Business-to-consumer (B2C) solutions, such as music streaming, photo sharing, and social network services.
- Enterprise-wide platform solutions, such as a shared Kubernetes cluster that's used by multiple business units within an organization.

When you build your own multitenant solution in Azure, there are several elements you need to consider that factor into your architecture.

In this series, we provide guidance about how to design, build, and operate your own multitenant solutions in Azure.

Note

In this series, we use the term *tenant* to refer to **your** tenants, which might be your customers or groups of users. Our guidance is intended to help you to build your own multitenant software solutions on top of the Azure platform.

Microsoft Entra ID also includes the concept of a tenant to refer to individual directories, and it uses the term *multitenancy* to refer to interactions between multiple Microsoft Entra tenants. Although the terms are the same, the concepts are not. When we need to refer to the Microsoft Entra concept of a tenant, we disambiguate it by using the full term *Microsoft Entra tenant*.

Scope

Azure is itself a multitenant service, and some of our guidance is based on our experience with running large multitenant solutions. However, the focus of this series is on helping you build your own multitenant services, while harnessing the power of the Azure platform.

Additionally, when you design a solution, there are many areas you need to consider. The content in this section is specific to how you design for multitenancy. We don't cover all of the features of the Azure services, or all of the architectural design considerations for every application. You should read this guide in conjunction with the [Microsoft Azure Well-Architected Framework](#) and the documentation for each Azure service that you use.

Intended audience

The guidance provided in this series is applicable to anyone building a multitenant application in Azure. The audience also includes anybody who is building SaaS products, such as independent software vendors (ISVs) and startups, whether those SaaS products are targeted for businesses or consumers. It also includes anyone building a product or platform that's intended to be used by multiple customers or tenants.

Some of the content in this series is designed to be useful for technical decision-makers, like chief technology officers (CTOs) and architects, and anyone designing or implementing a multitenant solution on Microsoft Azure. Other content is more technically focused and is targeted at solution architects and engineers who implement a multitenant solution.

Note

Managed service providers (MSPs) manage and operate Azure environments on behalf of their customers, and work with multiple Microsoft Entra tenants in the process. This is another form of multitenancy, but it's focused on managing Azure resources across multiple Microsoft Entra tenants. This series isn't intended to provide guidance on these matters.

However, the series is likely to be helpful for ISVs who build software for MSPs, or for anyone else who builds and deploys multitenant software.

What's in this series?

The content in this series is composed of three main sections:

- **Architectural considerations for a multitenant solution:** This section provides an overview of the key requirements and considerations you need to be aware of when planning and designing a multitenant solution.

The architectural considerations are particularly relevant for technical decision-makers, like chief technology officers (CTOs) and architects. Product managers will also find it valuable to understand how multitenancy affects their solutions. Additionally, anyone who works with multitenant architectures should have some familiarity with these principles and tradeoffs.

- **Architectural approaches for multitenancy:** This section describes the approaches you can consider when designing and building multitenant solutions, by using key cloud resource types. The section includes a discussion how to build multitenant solutions with compute, networking, storage, data, messaging, identity, AI/ML, and IoT components, as well as deployment, configuration, resource organization, governance, compliance, and cost management.

The architectural approaches are intended to be useful for solution architects and lead developers.

- **Service-specific guidance for a multitenant solution:** This section provides targeted guidance for specific Azure services. It includes discussions of the tenancy isolation models that you might consider for the components in your solution, as well as any features that are particularly relevant for a multitenant solution.

The service-specific guidance is useful for architects, lead developers, and anyone building or implementing Azure components for a multitenant solution.

Additionally, we provide a [list of related resources and links](#) for architects and developers of multitenant solutions.

Video

For an overview of the content covered in this series, and the basic concepts of multitenancy, see this video from Microsoft Reactor:

[https://www.youtube-nocookie.com/embed/aem8elgN7il ↗](https://www.youtube-nocookie.com/embed/aem8elgN7il)

Next steps

Review the [architectural considerations for a multitenant solution](#).

Architectural considerations for a multitenant solution

Azure

When you're considering a multitenant architecture, there are several decisions you need to make and elements you need to consider.

In a multitenant architecture, you share some or all of your resources between tenants. This process means that a multitenant architecture can give you cost and operational efficiency. However, multitenancy introduces complexities, including the following:

- How do you define what a *tenant* is, for your specific solution? Does a tenant correspond to a customer, a user, or a group of users (like a team)?
- How will you deploy your infrastructure to support multitenancy, and how much isolation will you have between tenants?
- What commercial pricing models will your solution offer, and how will your pricing models affect your multitenancy requirements?
- What level of service do you need to provide to your tenants? Consider performance, resiliency, security, and compliance requirements, like data residency.
- How do you plan to grow your business or solution, and will it scale to the number of tenants you expect?
- Do any of your tenants have unusual or special requirements? For example, does your biggest customer need higher performance or stronger guarantees than others?
- How will you monitor, manage, automate, scale, and govern your Azure environment, and how will multitenancy impact this?
- Which components of your solution handle tenant onboarding and management, and how should these components be designed?

Requirements

Whatever your architecture, it's essential that you have a clear understanding of your customers' or tenants' requirements. If you have made sales commitments to customers, or if you have contractual obligations or compliance requirements to meet, then you need to know what those requirements are when you architect your solution. But equally, your customers might have implicit expectations about how things *should* work, or how you *should* behave, which could affect the way you design a multitenant solution.

As an example, imagine you're building a multitenant solution that you sell to businesses in the financial services industry. Your customers have very strict security requirements, and they need you to provide a comprehensive list of every domain name that your solution uses, so they can add it to their firewall's allowlist. This requirement affects the Azure services you use and the level of isolation that you have to provide between your tenants. They also require that their solution has a minimum level of resiliency. There might be many similar expectations, both explicit and implicit, that you need to consider across your whole solution.

In this section, we outline the considerations that you should give, the requirements you should elicit, and some of the tradeoffs you need to make, when you are planning a multitenant architecture.

Intended audience

The articles in this section are particularly relevant for technical decision-makers, like chief technology officers (CTOs) and architects, as well as product managers. The audience also includes independent software vendors (ISVs) and startups who develop SaaS solutions. Additionally, anyone who works with multitenant architectures should have some familiarity with these principles and tradeoffs.

Next steps

Consider different [tenancy models](#) for your solution.

Tenancy models for a multitenant solution

Azure

There are many ways to consider how to work with tenants in your solution. Your choice of approach depends importantly on whether and how you share resources among your tenants. Intuitively, you might want to avoid sharing *any* resources, but that approach quickly becomes expensive as your business scales and you onboard more tenants.

When you consider the various models of multitenancy, it's helpful to first take into account how you define tenants for your organization, what your business drivers are, and how you plan to scale your solution. This article provides guidance to help technical decision makers evaluate the tenancy models and their tradeoffs.

Define a tenant

First, you need to define a *tenant* for your organization. Consider who your customer is. In other words, who are you providing your services to? There are two common models:

- **Business to business (B2B).** If your customers are other organizations, you're likely to map your tenants to those customers. However, consider whether your customers have divisions (teams or departments) and whether they have a presence in multiple countries/regions. You might need to map a single customer to multiple tenants if there are different requirements for these subgroups. Similarly, a customer might want to maintain two instances of your service so they can keep their development and production environments separated from each other. Generally, a single tenant has multiple users. For example, all of your customer's employees will be users within a single tenant.
- **Business to consumer (B2C).** If your customers are consumers, it's often more complicated to relate customers, tenants, and users. In some scenarios, each consumer might be a separate tenant. However, consider whether your solution might be used by families, groups of friends, clubs, associations, or other groups that might need to access and manage their data together. For example, a music streaming service might support both individual users and families, and it might treat each of these account types differently when it separates them into tenants.

Your definition of *tenant* will affect some of the things that you need to consider or emphasize when you architect your solution. For example, consider these types of

tenants:

- If your tenants are individual people or families, you might need to be particularly concerned about how you handle personal data, and the data sovereignty laws within each jurisdiction that you serve.
- If your tenants are businesses, you might need to be mindful of your customers' requirements for regulatory compliance, the isolation of their data, and ensuring that you meet a specified service-level objective (SLO), like uptime or service availability.

How do you decide which model to use?

Selecting a tenancy model isn't just a technical decision. It's also a commercial decision. You need to consider questions like these:

- What are your business objectives?
- Will your customers accept all forms of multitenancy? How does each multitenancy model affect your compliance requirements, or your customer's compliance requirements?
- Will a single-tenant solution scale to your future growth aspirations?
- How large is your operations team, and how much of your infrastructure management can you automate?
- Do your customers expect you to meet service-level agreements (SLAs), or do you have SLOs that you're aiming for?

If you expect your business to scale to a large number of customers, it's important to deploy shared infrastructure. Otherwise, you'll have to maintain a large and growing fleet of instances. Deploying individual Azure resources for each customer is likely to be unsustainable, unless you provision and use a dedicated subscription for each tenant. When you share the same Azure subscription across multiple tenants, [Azure resource quotas and limits](#) might start to apply, and the operational costs to deploy and reconfigure these resources increase with each new customer.

Conversely, if you expect that your business will have only a few customers, you might want to consider using single-tenant resources that are dedicated to each customer. Also, if your customers' isolation requirements are high, a single-tenant infrastructure might be appropriate.

Tenants and deployments

Next, you need to determine what *tenant* means for your particular solution, and whether you should distinguish between logical tenants and deployments.

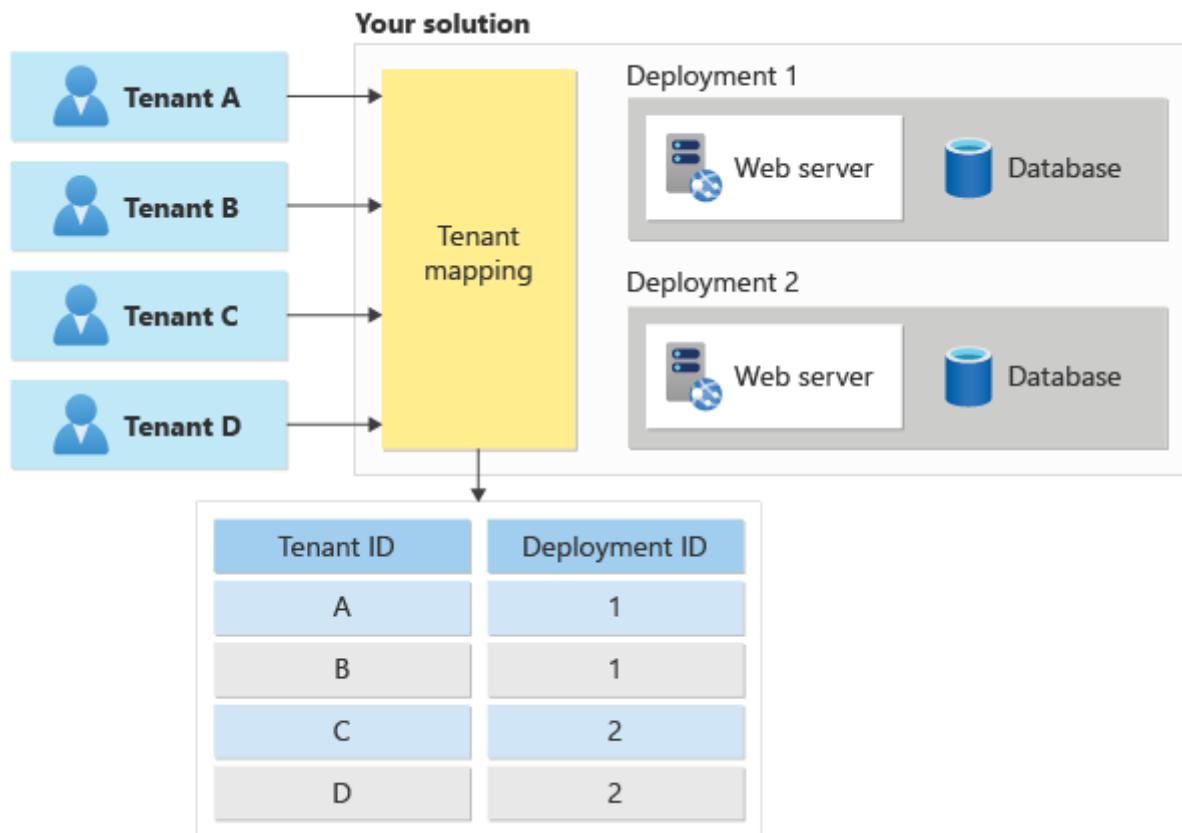
For example, consider a music streaming service. Initially, you might build a solution that can easily handle thousands (or even tens of thousands) of users. As you continue to grow, however, you might find that you need to duplicate your solution or some of its components in order to scale to new customer demand. This means that you need to determine how to assign specific customers to specific instances of your solution. You might assign customers randomly, or geographically, or by filling up a single instance and then starting another. However, you probably need to maintain a record of your customers and which infrastructure their data and applications are available on so that you can route their traffic to the correct infrastructure. In this example, you might represent each customer as a separate tenant, and then map the users to the deployment that contains their data. You have a one-to-many mapping between tenants and deployments, and you can move tenants among deployments at your own discretion.

In contrast, consider a company that creates cloud software for legal firms. Your customers might insist on having their own dedicated infrastructure to maintain their compliance standards. Therefore, you need to be prepared to deploy and manage many different instances of your solution from the start. In this example, a deployment always contains a single tenant, and a tenant is mapped to its own dedicated deployment.

A key difference between tenants and deployments is how isolation is enforced. When multiple tenants share a single deployment (a set of infrastructure), you typically rely on your application code and a tenant identifier that's in a database to keep each tenant's data separate. When you have tenants with their own dedicated deployments, they have their own infrastructure, so it might be less important for your code to be aware that it's operating in a multitenant environment.

Deployments are sometimes referred to as *supertenants* or *stamps*.

When you receive a request for a specific tenant, you need to map it to the deployment that holds that tenant's data, as shown here:

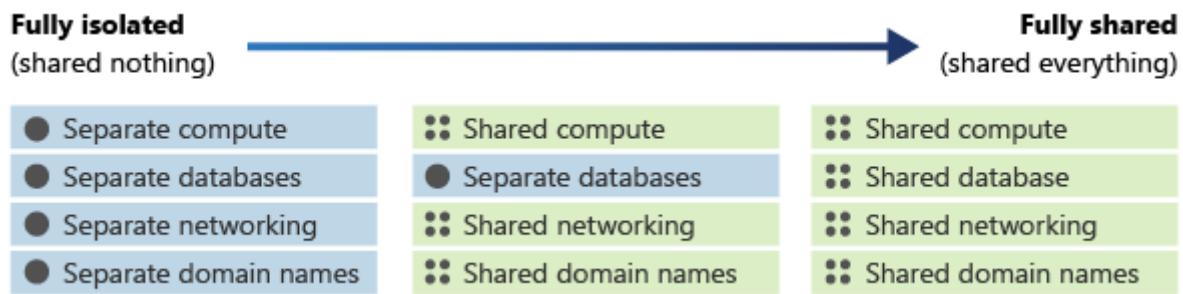


Tenant isolation

One of the biggest considerations in the design of a multitenant architecture is the level of isolation that each tenant needs. Isolation can mean different things:

- Having a single shared infrastructure, with separate instances of your application and separate databases for each tenant.
- Sharing some common resources, but keeping other resources separate for each tenant.
- Keeping data on a separate physical infrastructure. In the cloud, this configuration might require separate Azure resources for each tenant. It could even mean deploying a separate physical infrastructure by using [dedicated hosts](#).

Rather than thinking of isolation as a discrete property, you should think about it as being on a continuum. You can deploy components of your architecture that are more or less isolated than other components in the same architecture, depending on your requirements. The following diagram demonstrates a continuum of isolation:



The level of isolation affects many aspects of your architecture, including the following:

- **Security.** If you share infrastructure among multiple tenants, you need to be especially careful not to access data from one tenant when you return responses to another. You need a strong foundation for your identity strategy, and you need to consider both tenant and user identity in your authorization process.
- **Cost.** Shared infrastructure can be used by multiple tenants, so it's less expensive.
- **Performance.** If you share infrastructure, your system's performance might suffer as more customers use it, because the resources might be consumed faster.
- **Reliability.** If you use a single set of shared infrastructure, a problem with one tenant's components can result in an outage for everyone.
- **Responsiveness to individual tenant needs.** When you deploy infrastructure that's dedicated to one tenant, you might be able to adapt the configuration for the resources to that specific tenant's requirements. You might even consider this capability in your pricing model, allowing customers to pay more for isolated deployments.

Your solution architecture can influence your available options for isolation. For example, consider a three-tier solution architecture:

- Your user interface tier might be a shared multitenant web app, with all your tenants accessing a single host name.
- Your middle tier could be a shared application layer, with shared message queues.
- Your data tier could be isolated databases, tables, or blob containers.

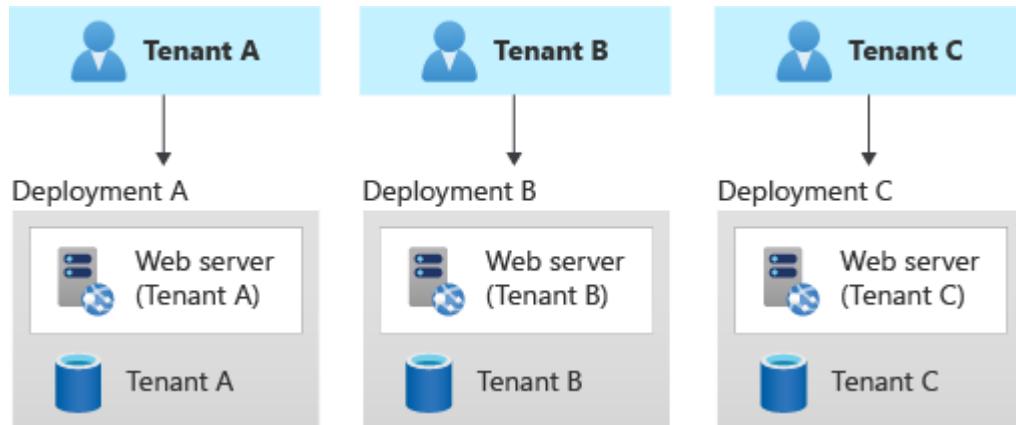
You can consider using different levels of isolation on each tier. You should base your decision about what's shared and what's isolated on many considerations, including cost, complexity, your customers' requirements, and the number of resources that you can deploy before reaching Azure quotas and limits.

Common tenancy models

After you establish your requirements, evaluate them against some common tenancy models and corresponding deployment patterns.

Automated single-tenant deployments

In an automated single-tenant deployment model, you deploy a dedicated set of infrastructure for each tenant, as shown in this example:



Your application is responsible for initiating and coordinating the deployment of each tenant's resources. Typically, solutions that use this model use infrastructure as code (IaC) or the Azure Resource Manager APIs extensively. You might use this approach when you need to provision entirely separate infrastructures for each of your customers. Consider the [Deployment Stamps pattern](#) when you plan your deployment.

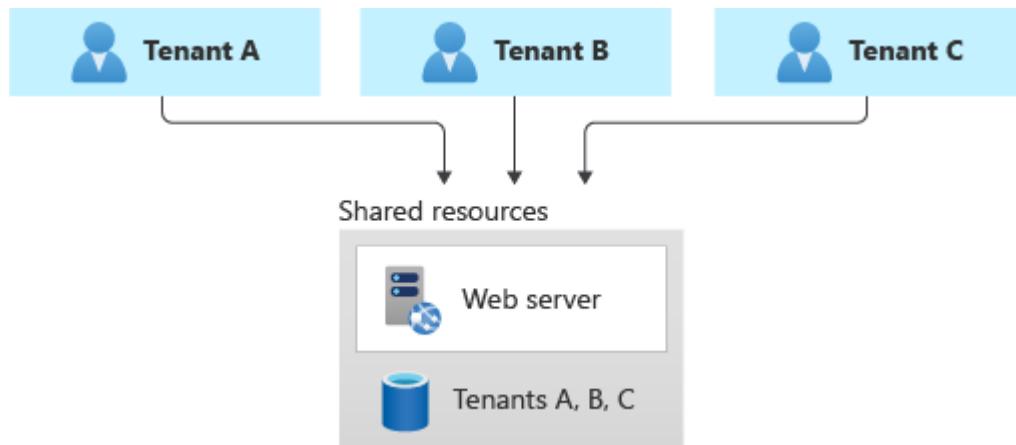
Benefits: A key benefit of this approach is that data for each tenant is isolated, which reduces the risk of accidental leakage. This safeguard can be important to some customers that have high regulatory compliance overhead. Additionally, tenants are unlikely to affect each other's system performance, an issue that's sometimes called the *noisy neighbor* problem. Updates and changes can be rolled out progressively across tenants, which reduces the likelihood of a system-wide outage.

Risks: If you use this approach, cost efficiency is low, because you don't share infrastructure among your tenants. If a single tenant requires a certain infrastructure cost, 100 tenants probably require 100 times that cost. Additionally, ongoing maintenance (like applying new configuration or software updates) will probably be time-consuming. Consider automating your operational processes, and consider applying changes progressively through your environments. You should also consider other cross-deployment operations, like reporting and analytics across your whole estate. Likewise, be sure to plan for how you can query and manipulate data across multiple deployments.

Fully multitenant deployments

At the opposite extreme, you can consider a fully multitenant deployment, where all components are shared. You have only one set of infrastructure to deploy and maintain,

and all tenants use it, as shown in the following diagram:



Benefits: This model is attractive because operating a solution that has shared components is less expensive. Even if you need to deploy higher tiers or SKUs of resources, the overall deployment cost is often still lower than the cost of a set of single-tenant resources. Additionally, if a user or tenant needs to move their data to another tenant, you might be able to update tenant identifiers and keys, and you might not have to migrate data between two separate deployments.

Risks:

- Be sure to separate data for each tenant, and don't leak data among tenants. You might need to manage sharding data. Additionally, you might need to be concerned about the effects that individual tenants can have on the overall system. For example, if a large tenant tries to perform a heavy query or operation, it might affect other tenants.
- Determine how to [track and associate your Azure costs to tenants](#), if doing so is important to you.
- Maintenance can be simpler with a single deployment, because you only have to update one set of resources. However, it's also often riskier, because changes might affect your entire customer base.
- You might also need to consider scale. You're more likely to reach [Azure resource scale limits](#) when you have a shared set of infrastructure. For example, if you use a storage account as part of your solution, as your scale increases, the number of requests to that storage account might reach the limit of what the storage account can handle. To avoid reaching a resource quota limit, you can consider deploying multiple instances of your resources (for example, multiple AKS clusters or storage accounts). You can even consider distributing your tenants across resources that you deploy into multiple Azure subscriptions.

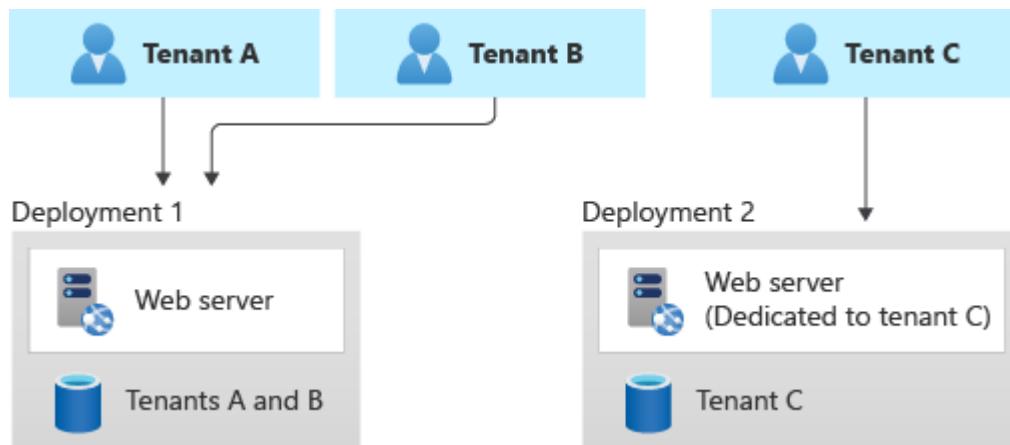
- There's probably a limit to how far you can scale a single deployment, and the costs of doing so might increase non-linearly. For example, if you have a single shared database, when you run at very high scale you might exhaust its throughput and need to pay increasingly more for increased throughput to keep up with your demand.

Vertically partitioned deployments

You don't have to choose one of the extremes of these scales. Instead, you can consider vertically partitioning your tenants by taking this approach:

- Use a combination of single-tenant and multitenant deployments. For example, you might have most of your customers' data and application tiers on multitenant infrastructures, but deploy single-tenant infrastructures for customers who require higher performance or data isolation.
- Deploy multiple instances of your solution geographically, and map each tenant to a specific deployment. This approach is particularly effective when you have tenants in different geographies.

Here's an example that illustrates a shared deployment for some tenants and a single-tenant deployment for another:



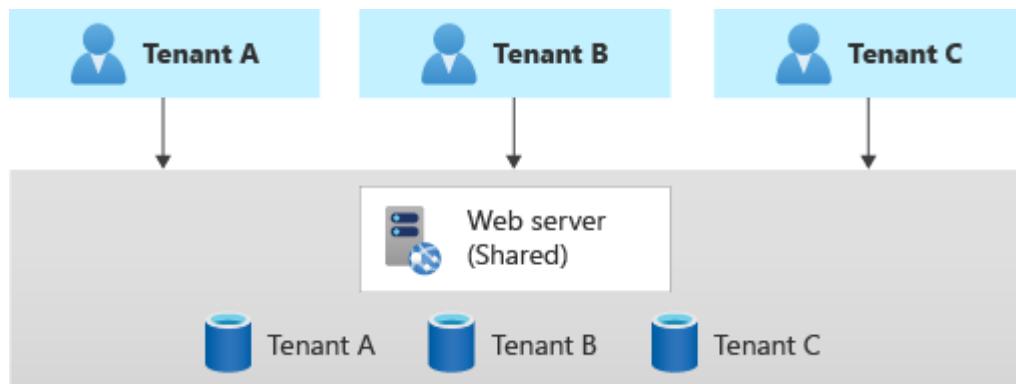
Benefits: Because you're still sharing infrastructure, you can gain some of the cost benefits of using shared multitenant deployments. You can deploy cheaper shared resources for certain customers, like customers who are evaluating your service with a trial. You can even bill customers a higher rate to use a single-tenant deployment, thereby recouping some of your costs.

Risks: Your codebase will probably need to be designed to support both multitenant and single-tenant deployments. If you plan to allow migration between infrastructures, you need to consider how to migrate customers from a multitenant deployment to their own single-tenant deployment. You also need to know which of your tenants are on

each deployment, so that you can communicate information about system issues or upgrades to the relevant customers.

Horizontally partitioned deployments

You can also consider horizontally partitioning your deployments. In a horizontal deployment, you have some shared components but maintain other components with single-tenant deployments. For example, you could build a single application tier and then deploy individual databases for each tenant, as shown in this diagram:



Benefits: Horizontally partitioned deployments can help you mitigate a noisy neighbor problem, if you identify that most of the load on your system is caused by specific components that you can deploy separately for each tenant. For example, your databases might absorb most of your system's load, because the query load is high. If a single tenant sends a large number of requests to your solution, the performance of a database might be negatively affected, but other tenants' databases (and shared components, like the application tier) remain unaffected.

Risks: With a horizontally partitioned deployment, you still need to consider the automated deployment and management of your components, especially the components used by a single tenant.

Test your isolation model

Whichever isolation model you choose, be sure to test your solution to verify that one tenant's data isn't accidentally leaked to another and that any [noisy neighbor](#) effects are acceptable. Consider using [Azure Chaos Studio](#) to deliberately introduce faults that simulate real-world outages and verify the resiliency of your solution even when components are malfunctioning.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Chad Kittel](#) | Principal Software Engineer
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Consider the [lifecycle of your tenants](#)

Tenant lifecycle considerations in a multitenant solution

Article • 03/02/2023

When you're considering a multitenant architecture, it's important to consider all of the different stages in a tenant's lifecycle. On this page, we provide guidance for technical decision-makers about the stages of the lifecycle and the important considerations for each stage.

Trial tenants

When you build a SaaS solution consider that many customers request or require trials before they commit to purchase a solution.

Trials bring along the following unique considerations:

- **Service requirements:** Should trials be subject to the same data security, performance, and service-level requirements as the data for full customers?
- **Infrastructure:** Should you use the same infrastructure for trial tenants as for full customers, or should you have dedicated infrastructure for trial tenants?
- **Migration:** If customers purchase your service after a trial, how will they migrate the data from their trial tenants into their paid tenants?
- **Request process:** Are there limits around who can request a trial? How can you prevent abuse of your solution? Do you allow automated creation of trial tenants or does your team get involved in each request?
- **Limits:** What limits do you want or need to place on trial customers, such as time limits, feature restrictions, or limitations around performance?

In some situations, a [freemium pricing model](#) can be an alternative to providing trials.

Onboard new tenants

When onboarding a new tenant, consider the following:

- **Process:** Will onboarding be a self-service, automated, or manual process?
- **Data residency:** Does the tenant have any specific requirements for data residency? For example, are there data sovereignty regulations in effect?
- **Compliance:** Does the tenant have to meet any compliance standards (such as PCI DSS, HIPAA, and so on)?

- **Disaster recovery:** Does the tenant have any specific disaster recovery requirements, such as a recovery time objective (RTO) or a recovery point objective (RPO)? Are these different from the guarantees that you provide to other tenants?
- **Information:** What information do you require, to be able to fully onboard the tenant? For example, do you need to know their organization's legal name? Do you need their company logo to brand the application, and if so, what file size and format do you need?
- **Billing:** Does the platform provide different pricing options and billing models?
- **Environments:** Does the tenant require pre-production environments? And are there set expectations on availability for that environment? Is it transient (on-demand) or always available?

After tenants have been onboarded, they move into a 'business as usual' state. However, there are still several important lifecycle events that can occur, even when they are in this state.

Update tenants' infrastructure

You will need to consider how you apply updates to your tenants' infrastructure. Different tenants might have updates applied at different times.

See [Updates](#) for other considerations about updating tenants' deployments.

Scale tenants' infrastructure

Consider whether your tenants might have seasonal business patterns, or otherwise change the level of consumption for your solution.

For example, if you provide a solution to retailers, you might expect that certain times of the year will be particularly busy in some geographic regions, and quiet at other times. Consider whether this seasonality affects the way you design and scale your solution. Be aware of how seasonality might affect [noisy neighbor issues](#), such as when a subset of tenants experience a sudden and unexpected increase in load that reduces the performance of other tenants. You can consider applying mitigations, which might include scaling individual tenants' infrastructure, moving tenants between deployments, and provisioning a sufficient level of capacity to handle spikes and troughs in traffic.

Move tenants between infrastructure

You might need to move tenants between infrastructure for a number of reasons, including the following:

- **Rebalancing:** You follow a [vertically partitioned approach](#) to map your tenants to infrastructure, and you need to move a tenant to a different deployment in order to rebalance your load.
- **Upgrades:** A tenant upgrades their SKU or pricing tier, and they need to be moved to a single-tenant, dedicated deployment with higher isolation from other tenants.
- **Migrations:** A tenant requests their data be moved to a dedicated data store.
- **Region moves:** A tenant requires their data be moved to a new geographic region. This might occur during a company acquisition, or when laws or geopolitical situations change.

Consider how you move your tenants' data, as well as redirect requests to the new set of infrastructure that hosts their instance. You should also consider whether moving a tenant might result in downtime, and make sure tenants are fully aware of the risk.

Merge and split tenants

It's tempting to think of tenants or customers as static, unchanging entities. However, in reality, this often isn't true. For example:

- In business scenarios, companies might be acquired or merge, including companies located in different geographic regions.
- Similarly, in business scenarios, companies might split or divest.
- In consumer scenarios, individual users might join or leave families.

Consider whether you need to provide capabilities to manage the merging and separation of data, user identities, and resources. Also, consider how data ownership affects your handling of merge and split operations. For example, consider a consumer photography application built for families to share photos with one another. Are the photos owned by the individual family members who contributed them, or by the family as a whole? If users leave the family, should their data be removed or remain in the family's data set? If users join another family, should their old photos move with them?

Offboard tenants

It's also inevitable that tenants will occasionally need to be removed from your solution. In a multitenant solution, this brings along some important considerations, including the following:

- **Retention period:** How long should you maintain the customer data? Are there legal requirements to destroy data, after a certain period of time?
- **Re-onboarding:** Should you provide the ability for customers to be re-onboarded?

- **Rebalancing:** If you run shared infrastructure, do you need to rebalance the allocation of tenants to infrastructure?

Deactivate and reactivate tenants

There are situations where a customer's account might need to be deactivated or reactivated. For example:

- The customer has requested deactivation. In a consumer system, a customer might opt to unsubscribe.
- The customer can't be billed, and you need to deactivate the subscription.

Deactivation is separate to offboarding in that it's intended to be a temporary state. However, after a period of time, you might choose to offboard a deactivated tenant.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Chad Kittel](#) | Principal Software Engineer
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Consider the [pricing models](#) you will use for your solution.

Pricing models for a multitenant solution

Article • 12/14/2023

A good pricing model ensures that you remain profitable as the number of tenants grows and as you add new features. An important consideration when developing a commercial multitenant solution is how to design pricing models for your product. On this page, we provide guidance for technical decision-makers about the pricing models you can consider and the tradeoffs involved.

When you determine the pricing model for your product, you need to balance the *return on value* (ROV) for your customers with the *cost of goods sold* (COGS) to deliver the service. Offering more flexible commercial models (for a solution) might increase the ROV for customers, but it might also increase the architectural and commercial complexity of the solution (and therefore also increase your COGS).

Some important considerations that you should take into account, when developing pricing models for a solution, are as follows:

- Will the COGS be higher than the profit you earn from the solution?
- Can the COGS change over time, based on growth in users or changes in usage patterns?
- How difficult is it to [measure and record the information required to operate the pricing model](#)? For example, if you plan to bill your customers based on the number of API calls they make, have you identified how you measure the API calls made by each customer?
- Does your profitability depend on ensuring customers use your solution in a limited way?
- If a customer overuses the solution, does that mean you're no longer profitable?

There are some key factors that influence your profitability:

- **Azure service pricing models.** The pricing models of the Azure or third-party services that make up your solution may affect which models are profitable.
- **Service usage patterns.** Users may only need to access your solution during their working hours or may only have a small percentage of high-volume users. Can you reduce your COGS by reducing the unused capacity when your usage is low?
- **Storage growth.** Most solutions accumulate data over time. More data means a higher cost to store and protect it, reducing your profitability per tenant. Can you set storage quotas or enforce a data retention period?

- **Tenant isolation.** The [tenancy model](#) you use affects the level of isolation you have between your tenants. If you share your resources, do you need to consider how tenants might over-utilize or abuse the service? How will the level of tenant isolation affect your COGS and performance for everyone? Some pricing models aren't profitable without additional controls around resource allocation. For example, you might need to implement service throttling to make a flat-rate pricing model sustainable.
- **Tenant lifecycle.** For example, solutions with high customer churn rates, or services that require a greater on-boarding effort, may suffer lower profitability--especially if they're priced using a consumption-based model.
- **Service level requirements.** Tenants that require higher levels of service may mean your solution isn't profitable anymore. It's critical that you're clear about your customers' service-level expectations and any obligations you have, so that you can plan your pricing models accordingly.

Common pricing models

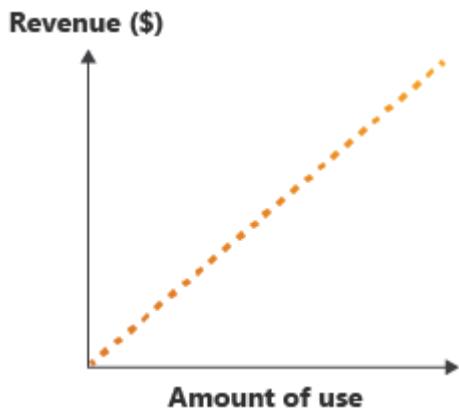
There are many common pricing models that are often used with multitenant solutions. Each of these pricing models has associated commercial benefits and risks, and requires additional architectural considerations. It's important to understand the differences between these pricing models, so that you can ensure your solution remains profitable as it evolves.

Note

You can offer multiple models for a solution or combine models together. For example, you could provide a per-user model for your customers that have fairly stable user numbers, and you can also offer a consumption model for customers who have fluctuating usage patterns.

Consumption-based pricing

A consumption model is sometimes referred to as *pay-as-you-go*, or *PAYG*. As the use of your service increases, your revenue increases:



When you measure consumption, you can consider simple factors, such as the amount of data being added to the solution. Alternatively, you might consider a combination of usage attributes together. Consumption models offer many benefits, but they can be difficult to implement in a multitenant solution.

Benefits: From your customers' perspective, there's minimal upfront investment that's required to use your solution, so that this model has a low barrier to entry. From your perspective as the service operator, your hosting and management costs increase as your customers' usage and your revenue increases. This increase can make it a highly scalable pricing model. Consumption pricing models work especially well when the Azure services that are used in the solution are consumption-based too.

Complexity and operational cost: Consumption models rely on accurate measurements of usage and on splitting this usage by tenant. This can be challenging, especially in a solution with many distributed components. You need to keep detailed consumption records for billing and auditing as well as providing methods for customers to get access to their consumption data.

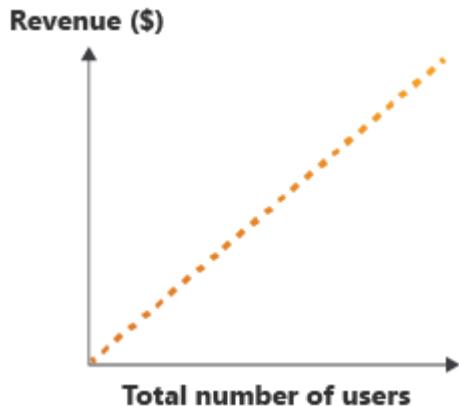
Risks: Consumption pricing can motivate your customers to reduce their usage of your system, in order to reduce their costs. Additionally, consumption models result in unpredictable revenue streams. You can mitigate this by offering *capacity reservations*, where customers pay for some level of consumption upfront. You, as the service provider, can use this revenue to invest in improvements in the solution, to reduce the operational cost or to increase the return on value by adding features.

ⓘ Note

Implementing and supporting capacity reservations may increase the complexity of the billing processes within your application. You might also need to consider how customers get refunds or exchange their capacity reservations, and these processes can also add commercial and operational challenges.

Per-user pricing

A per-user pricing model involves charging your customers based on the number of people using your service, as demonstrated in the following diagram.



Per-user pricing models are very common, due to their simplicity to implement in a multitenant solution. However, they're associated with several commercial risks.

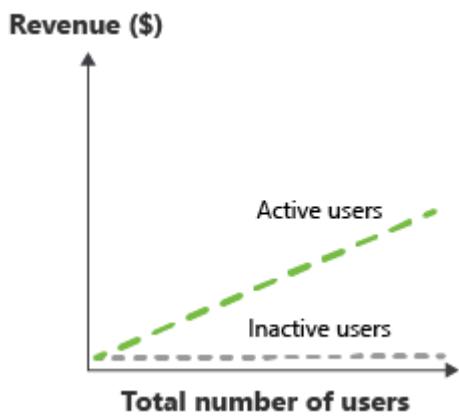
Benefits: When you bill your customers for each user, it's easy to calculate and forecast your revenue stream. Additionally, assuming that you have fairly consistent usage patterns for each user, then revenue increases at the same rate as service adoption, making this a scalable model.

Complexity and operational cost: Per-user models tend to be easy to implement. However, in some situations, you need to measure per-user consumption, which can help you to ensure that the COGS for a single user remains profitable. By measuring the consumption and associating it with a particular user, you can increase the operational complexity of your solution.

Risks: Different user consumption patterns might result in a reduced profitability. For example, heavy users of the solution might cost more to serve than light users. Additionally, the actual return on value (ROV) for the solution isn't reflected by the actual number of user licenses purchased.

Per-active user pricing

This model is similar to [per-user pricing](#), but rather than requiring an upfront commitment from the customer on the number of expected users, the customer is only charged for users that actually log into and use the solution over a period (as shown in the following diagram).



You can measure this in whatever period makes sense. Monthly periods are common, and then this metric is often recorded as *monthly active users* or *MAU*.

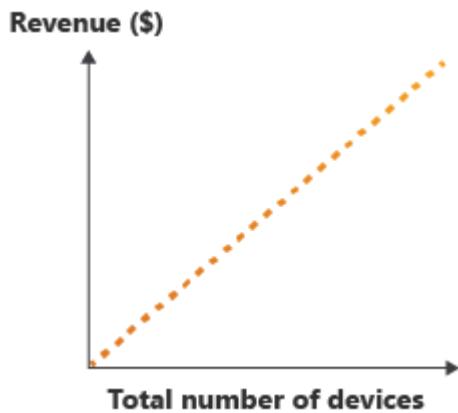
Benefits: From your customers' perspective, this model requires a low investment and risk, because there's minimal waste; unused licenses aren't billable. This makes it particularly attractive when marketing the solution or growing the solution for larger enterprise customers. From your perspective as the service owner, your ROV is more accurately reflected to the customer by the number of monthly active users.

Complexity and operational cost: Per-active user models require you to record actual usage, and to make it available to a customer as part of the invoice. Measuring per-user consumption helps to ensure profitability is maintained with the COGS for a single user, but again it requires additional work to measure the consumption for each user.

Risks: Like per-user pricing, there's a risk that the different consumption patterns of individual users may affect your profitability. Compared to per-user pricing, per-active user models have a less predictable revenue stream. Additionally, [discount pricing](#) doesn't provide a useful way of stimulating growth.

Per-unit pricing

In many systems, the number of users isn't the element that has the greatest effect on the overall COGS. For example, in device-oriented solutions, also referred to as the *internet of things* or *IoT*, the number of devices often has the greatest impact on COGS. In these systems, a per-unit pricing model can be used, where you define what a *unit* is, such as a device. See the following diagram.



Additionally, some solutions have highly variable usage patterns, where a small number of users has a disproportionate impact on the COGS. For example, in a solution sold to brick-and-mortar retailers, a per-store pricing model might be appropriate.

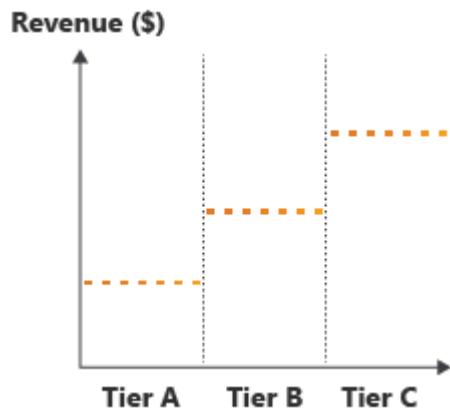
Benefits: In systems where individual users don't have a significant effect on COGS, per-unit pricing is a better way to represent the reality of how the system scales and the resulting impact to COGS. It also can improve the alignment to the actual patterns of usage for a customer. For many IoT solutions, where each device generates a predictable and constant amount of consumption, this can be an effective model to scale your solution's growth.

Complexity and operational cost: Generally, per-unit pricing is easy to implement and has a fairly low operational cost. However, the operational cost can become higher if you need to differentiate and track usage by individual units, such as devices or retail stores. Measuring per-unit consumption helps you ensure your profitability is maintained, since you can determine the COGS for a single unit.

Risks: The risks of a per-unit pricing model are similar to per-user pricing. Different consumption patterns by some units may mean that you have reduced profitability, such as if some devices or stores are much heavier users of the solution than others.

Feature- and service-level based pricing

You may choose to offer your solution with different tiers of functionality at different price points. For example, you might provide two monthly flat-rate or per-unit prices, one being a basic offering with a subset of features available, and the other presenting the comprehensive set of your solution's features. See the following diagram.



This model may also offer different service-level agreements for different tiers. For example, your basic tier may offer 99.9% uptime, whereas a premium tier may offer 99.99%. The higher service-level agreement (SLA) could be implemented by using services and features that enable higher [availability targets](#).

Although this model can be commercially beneficial, it does require mature engineering practices to do well. With careful consideration, this model can be very effective.

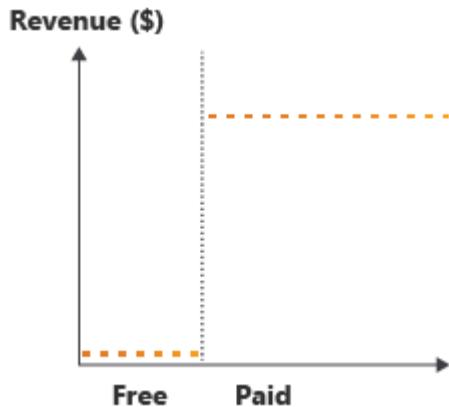
Benefits: Feature-based pricing is often attractive to customers, since they can select a tier based on the feature set or service level they need. It also provides you with a clear path to upsell your customers with additional features or higher resiliency for those who require it.

Complexity and operational cost: Feature-based pricing models can be complex to implement, since they require your solution to be aware of the features that are available at each price tier. Feature toggles can be an effective way to provide access to certain subsets of functionality, but this requires ongoing maintenance. Also, feature toggles increase the overhead to ensure high quality, because there will be more code paths to test. Enabling higher service availability targets in some tiers may require additional architectural complexity, to ensure the right set of infrastructure is used for each tier, and this process may increase the operational cost of the solution.

Risks: Feature-based pricing models can become complicated and confusing, if there are too many tiers or options. Additionally, the overhead involved in dynamically toggling features can slow down the rate at which you deliver additional features.

Freemium pricing

You might choose to offer a free tier of your service, with basic functionality and no service-level guarantees. You then might offer a separate paid tier, with additional features and a formal service-level agreement (as shown in the following diagram).



The free tier may also be offered as a time-limited trial, and during the trial your customers might have full or limited functionality available. This is referred to as a freemium model, which is effectively an extension of the [feature-based pricing model](#).

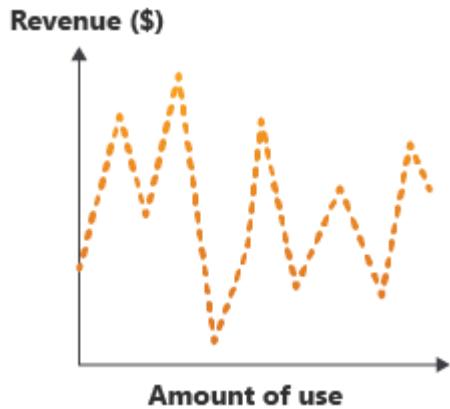
Benefits: It's very easy to market a solution when it's free.

Complexity and operational cost: All of the complexity and operational cost concerns apply from the feature-based pricing model. However, you also have to consider the operational cost involved in managing free tenants. You might need to ensure that stale tenants are offboarded or removed, and you must have a clear retention policy, especially for free tenants. When promoting a tenant to a paid tier, you might need to move the tenant between Azure services, to obtain higher SLAs. It will also be important to retain the tenant's data and configuration, when moving to a paid tier.

Risks: You need to ensure that you provide a high enough ROV for tenants to consider switching to a paid tier. Additionally, the cost of providing your solution to customers on the free tier needs to be covered by the profit margin from those who are on paid tiers.

Cost of goods sold pricing

You might choose to price your solution so that each tenant only pays the cost of operating their share of the Azure services, with no added profit margin. This model - also called *pass through cost or pricing* - is sometimes used for multitenant solutions that aren't intended to be a profit center.



The cost of goods sold model is a good fit for internally facing multitenant solutions. Each organizational unit corresponds to a tenant, and the costs of your Azure resources need to be spread between them. It might also be appropriate where revenue is derived from sales of other products and services that consume or augment the multitenant solution.

Benefits: Because this model doesn't include any added margin for profit, the cost to tenants will be lower.

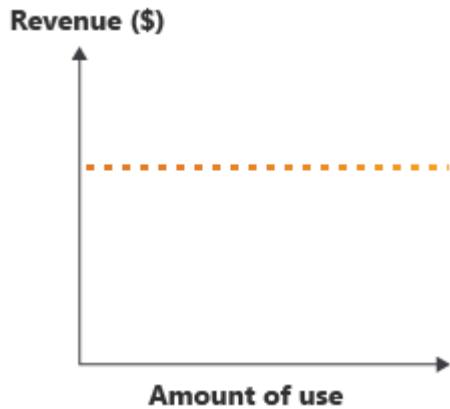
Complexity and operational cost: Similar to the consumption model, cost of goods sold pricing relies on [accurate measurements of usage](#) and on splitting this usage by tenant. Tracking consumption can be challenging, especially in a solution with many distributed components. You need to keep detailed consumption records for billing and auditing as well as providing methods for customers to get access to their consumption data.

For internally facing multitenant solutions, tenants might accept approximate cost estimates and have more relaxed billing audit requirements. These relaxed requirements reduce the complexity and cost of operating your solution.

Risks: Cost of goods sold pricing can motivate your tenants to reduce their usage of your system, in order to reduce their costs. However, because this model is used for applications that aren't profit centers, this might not be a concern.

Flat-rate pricing

In this model, you charge a flat rate to a tenant for access to your solution, for a given period of time. The same pricing applies regardless of how much they use the service, the number of users, the number of devices they connect, or any other metric. See the following diagram.



This is the simplest model to implement and for customers to understand, and it's often requested by enterprise customers. However, it can easily become unprofitable if you need to continue to add new features or if tenant consumption increases without any additional revenue.

Benefits: Flat-rate pricing is easy to sell, and it's easy for your customers to understand and budget for.

Complexity and operational cost: Flat-rate pricing models can be easy to implement because billing customers doesn't require any metering or tracking consumption. However, while not essential, it's advisable to measure per-tenant consumption to ensure that you're measuring COGS accurately and to ensure that your profitability is maintained.

Risks: If you have tenants who make heavy use of your solution, then it's easy for this pricing model to become unprofitable.

Discount pricing

Once you've defined your pricing model, you might choose to implement commercial strategies to incentivize growth through discount pricing. *Discount pricing* can be used with consumption, per-user, and per-unit pricing models.

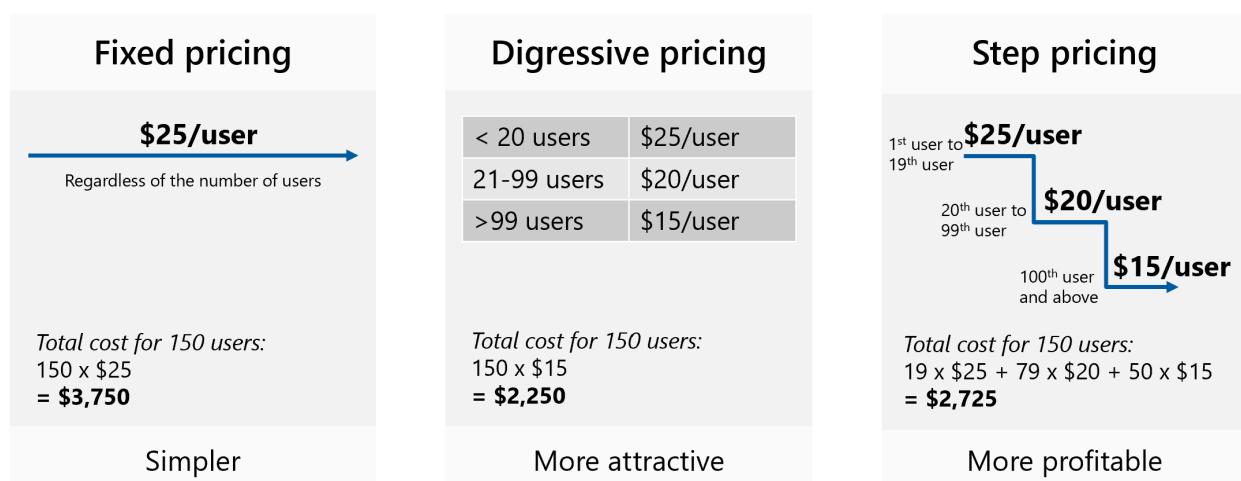
ⓘ Note

Discount pricing doesn't typically require architectural considerations, beyond adding support for a more complex billing structure. A complete discussion into the commercial benefits of discounting is beyond the scope of this document.

Common discount pricing patterns include:

- **Fixed pricing.** You have the same cost for each user, unit, or amount of consumption, no matter how much is purchased or consumed. This is the simplest approach. However, customers who make heavy use of your solution may feel like they should benefit from economies of scale by getting a discount.
- **Digressive pricing.** As customers purchase or consume more units, you reduce the cost per unit. This is more commercially attractive to customers.
- **Step pricing.** You reduce the cost per unit, as customers purchase more. However, you do so in step changes, based on predefined ranges of quantity. For example, you might charge a higher price for the first 100 users, then a lower price for the 101st to 200th user, then a lower price again after that. This can be more profitable.

The following diagram illustrates these pricing patterns.



Non-production environment discounts

In many cases, customers require access to a non-production environment that they can use for testing, training, or for creating their own internal documentation. Non-production environments usually have lower consumption requirements and costs to operate. For example, non-production environments often aren't subject to service-level agreements (SLAs), and [rate limits](#) might be set at lower values. You might also consider more aggressive [autoscaling](#) on your Azure services.

Equally, customers often expect non-production environments to be significantly cheaper than their production environments. There are several alternatives that might be appropriate, when you provide non-production environments:

- Offer a [freemium tier](#), like you might already do for paid customers. This should be carefully monitored, as some organizations might create many testing and training environments, which will consume additional resources to operate.

ⓘ Note

Time-limited trials using freemium tiers aren't usually suitable for testing and training environments. Customers usually need their non-production environments to be available for the lifetime of the production service.

- Offer a testing or training tier of your service, with [lower usage limits](#). You may choose to restrict the availability of this tier to customers who have an existing paid tenant.
- Offer a discounted [per-user](#), [per-active user](#), or [per-unit](#) pricing for non-production tenants, with a lower or no service level agreement.
- For tenants using [flat-rate pricing](#), non-production environments might be negotiated as part of the agreement.

ⓘ Note

Feature-based pricing is not usually a good option for non-production environments, unless the features offered are the same as what the production environment offers. This is because tenants will usually want to test and provide training on all the same features that are available to them in production.

Unprofitable pricing models

An unprofitable pricing model costs you more to deliver the service than the revenue you earn. For example, you might charge a flat rate per tenant without any limits for your service, but then you would build your service with consumption-based Azure resources and without per-tenant [usage limits](#). In this situation, you may be at risk of your tenants overusing your service and, thus, making it unprofitable to serve them.

Generally, you want to avoid unprofitable pricing models. However, there are situations where you might choose to adopt an unprofitable pricing model, including:

- A free service is being offered to enable growth.
- Additional revenue streams are provided by services or add-on features.
- Hosting a specific tenant provides another commercial value, such as using them as an anchor tenant in a new market.

In the case that you inadvertently create an unprofitable pricing model, there are some ways to mitigate the risks associated with them, including:

- Limit the use of the service through [usage limits](#).

- Require the use of capacity reservations.
- Request the tenant move to a higher feature or service tier.

Risky pricing models

When implementing a pricing model for a solution, you'll usually have to make assumptions about how it will be used. If these assumptions turn out to be incorrect or the usage patterns change over time, then your pricing model may become unprofitable. Pricing models that are at risk of becoming unprofitable are known as *risky pricing* models. For example, if you adopt a pricing model that expects users to self-limit the amount that they use your solution, then you may have implemented a risky pricing model.

Most SaaS solutions will add new features regularly. This increases the ROV to users, which may also lead to increases in the amount the solution is used. This could result in a solution that becomes unprofitable, if the use of new features drives usage, but the pricing model doesn't factor this in.

For example, if you introduce a new video upload feature to your solution, which uses a consumption-based resource, and user uptake of the feature is higher than expected, then consider a combination of [usage limits](#) and [feature and service level pricing](#).

Usage limits

Usage limits enable you to restrict the usage of your service in order to prevent your pricing models from becoming unprofitable, or to prevent a single tenant from consuming a disproportionate amount of the capacity of your service. This can be especially important in multitenant services, where a single tenant can impact the experience of other tenants by over-consuming resources.

ⓘ Note

It's important to make your customers aware that you apply usage limits. If you implement usage limits without making your customers aware of the limit, then it will result in customer dissatisfaction. This means that it's important to identify and plan usage limits ahead of time. The goal should be to plan for the limit, and to then communicate the limits to customers before they become necessary.

Usage limits are often used in combination with [feature and service-level pricing](#), to provide a higher amount of usage at higher tiers. Limits are also commonly used to

protect core components that will cause system bottlenecks or performance issues, if they're over-consumed.

Rate limits

A common way to apply a usage limit is to add rate limits to APIs or to specific application functions. This is also referred to as [throttling](#). Rate limits prevent continuous overuse. They're often used to limit the number of calls to an API, over a defined time period. For example, an API may only be called 20 times per minute, and it will return an HTTP 429 error, if it is called more frequently than this.

Some situations, where rate limiting is often used, include the following:

- REST APIs.
- Asynchronous tasks.
- Tasks that aren't time sensitive.
- Actions that incur a high cost to execute.
- Report generation.

Implementing rate limiting can increase the solution complexity, but services like Azure API Management can make this simpler by applying [rate limit policies](#).

Pricing model lifecycle

Like any other part of your solution, pricing models have a lifecycle. As your application evolves over time, you might need to change your pricing models. This may be driven by changing customer needs, commercial requirements, or changes to functionality within your application. Some common pricing lifecycle changes include the following:

- Adding a completely new pricing model. For example, adding a [consumption pricing model](#) to a solution that currently offers a [flat rate model](#).
- Retiring an existing pricing model.
- Adding a tier to an existing pricing model.
- Retiring a tier in an existing pricing model.
- Changing usage limits on a feature in a pricing model.
- Adding or removing features from a [feature and service-level pricing model](#).
- Changing from a business-to-consumer (B2C) commercial model, to a business-to-business (B2B) commercial model. This change then necessitates new pricing structures for your business customers.

It is usually complex to implement and manage many different pricing models at once. It's also confusing to your customers. So, it's better to implement only one or two

models, with a small number of tiers. This makes your solution more accessible and easier to manage.

ⓘ Note

Pricing models and billing functions should be tested, ideally using automated testing, just like any other part of your system. The more complex the pricing models, the more testing is required, and so the cost of development and new features will increase.

When changing pricing models, you'll need to consider the following factors:

- Will tenants want to migrate to the new model?
- Is it easy for tenants to migrate to the new model?
- Will new pricing models expose your service to risk? For example, are you removing rate limits that are currently protecting critical resources from overuse?
- Do tenants have a clear path for migrating to the new pricing model?
- How will you prevent tenants from using older pricing models, so that you can retire them?
- Are tenants likely to receive *bill shock* (a negative reaction to an unexpected bill) upon changing pricing models?
- Are you monitoring the performance and utilization of your services, for new or changed pricing models, so that you can ensure continued profitability?
- Are you able to clearly communicate the ROV for new pricing models, to your existing tenants?

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Daniel Scott-Raynsford](#) | Partner Technology Strategist

Other contributors:

- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure
- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Chad Kittel](#) | Principal Software Engineer
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirsksiy](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

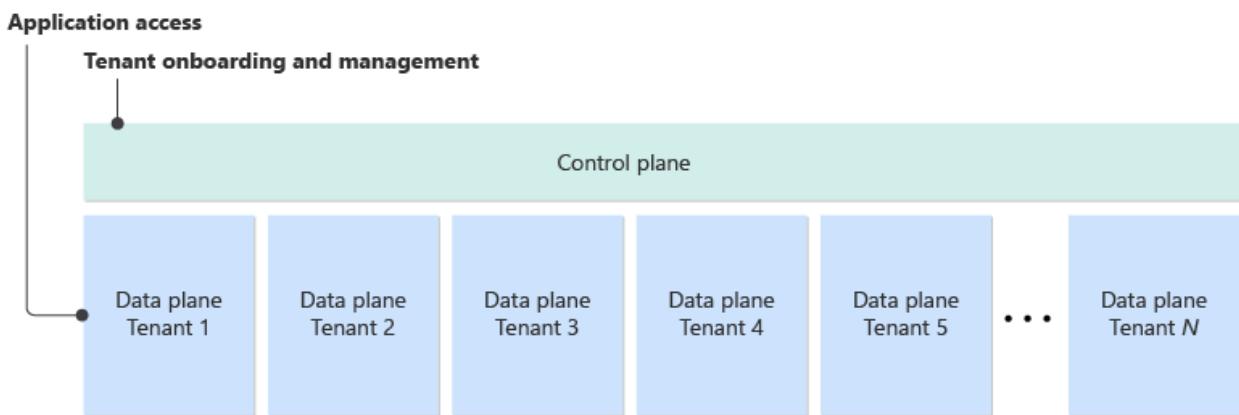
Consider how you'll [measure consumption](#) by tenants in your solution.

Considerations for multitenant control planes

Azure

A multitenant solution has multiple *planes*, and each has its own responsibilities. The *data plane* enables end users and clients to interact with the system. The *control plane* is the component that manages higher-level tasks across all tenants, like access control, provisioning, and system maintenance to support your platform administrators' tasks.

This article provides information about the responsibilities of control planes and how to design a control plane that meets your needs.



For example, consider a bookkeeping system for managing financial records. Multiple tenants store their financial records in the system. When end users access the system to view and enter their financial records, they use the data plane. The data plane is likely the primary application component for your solution. Your tenants probably think of it as the way to use the system for its intended purpose. The control plane is the component that onboards new tenants, creates databases for each tenant, and performs other management and maintenance operations. If the system didn't have a control plane, administrators would need to run many manual processes. Or data plane and control plane tasks would be mixed together, overcomplicating the solution.

Many complex systems include control planes. For example, the Azure control plane, [Azure Resource Manager](#), is a set of APIs, tools, and back-end components that are responsible for deploying and configuring Azure resources. The [Kubernetes control plane](#) ↗ manages many tasks, like the placement of Kubernetes pods on worker nodes. Almost all software as a service (SaaS) solutions have a control plane to handle cross-tenant tasks.

When you design multitenant solutions, you need to consider control planes. The following sections provide the details you need to scope and design a control plane.

Responsibilities of a control plane

There's no single template for a control plane or its responsibilities. Your solution's requirements and architecture dictate what your control plane needs to do. In some multitenant solutions, the control plane has a wide range of responsibilities and is a complex system in its own right. In other multitenant solutions, the control plane has only basic responsibilities.

In general, a control plane might have many of the following core responsibilities:

- Provision and manage the system resources that the system needs to serve the workload, including tenant-specific resources. Your control plane might [invoke and orchestrate a deployment pipeline](#) that's responsible for deployments, or it might run the deployment operations itself.
- [Reconfigure shared resources](#) to be aware of new tenants. For example, your control plane might configure network routing to ensure that incoming traffic is [mapped to the correct tenant's resources](#), or you might need to scale your resource capacity.
- Store and manage the configuration of each tenant.
- Handle [tenant lifecycle events](#), including onboarding, moving, and offboarding tenants.
- Track each tenant's use of your features and the performance of the system.
- [Measure each tenant's consumption](#) of your system's resources. Consumption metrics might inform your billing systems, or they might be used for resource governance.

If you use the [fully multitenant tenancy model](#) and don't deploy any tenant-specific resources, a basic control plane might just track tenants and their associated metadata. For example, whenever a new tenant signs up to your service, the control plane could update the appropriate records in a database so that the rest of the system is able to serve the new tenant's requests.

In contrast, suppose your solution uses a deployment model that requires tenant-specific infrastructure, like the [automated single-tenant model](#). In this scenario, your control plane might have additional responsibilities, like deploying or reconfiguring Azure infrastructure whenever you onboard a new tenant. Your solution's control plane probably needs to interact with the control planes for the services and technologies that you use, like Azure Resource Manager or the Kubernetes control plane.

More advanced control planes might also take on more responsibilities:

- Perform automated maintenance operations. Common maintenance operations include deleting or archiving old data, creating and managing database indexes, and rotating secrets and cryptographic certificates.
- Allocate tenants to existing deployments or stamps, which is sometimes called *tenant placement*.
- Rebalance existing tenants across deployment stamps.
- Integrate with external customer management solutions, like Microsoft Dynamics 365, to track customer activity.

Scope a control plane

You need to carefully consider how much effort to spend on building a control plane for your solution. Control planes by themselves don't provide immediate customer value, so it might not be easy to justify spending engineering effort on designing and building a high-quality control plane. However, as your system grows and scales, you'll increasingly need automated management and operations to be able to keep up with your growth.

In certain situations, you might not need a full control plane. This situation might apply if your system will have fewer than five tenants. Instead, your team can take on a control plane's responsibilities and can use manual operations and processes to onboard and manage tenants. However, you should still have a process and central place to track your tenants and their configurations.

Tip

If you decide not to create a full control plane, it's still a good idea to be systematic about your management procedures:

- Document your processes thoroughly.
- Wherever possible, create and reuse scripts for your management operations.

If you need to automate the processes in the future, your documentation and scripts can form the basis of your control plane.

As you grow beyond a few tenants, you'll likely benefit from having a way to track each tenant and apply monitoring across your fleet of resources and tenants. You might also notice that your team spends an increasing amount of time and effort on tenant management. Or you might notice bugs or operational problems because of inconsistencies in the ways that team members perform management tasks. If these

situations occur, it's worth considering building a more comprehensive control plane to take on these responsibilities.

ⓘ Note

If you provide self-service tenant management, you need a control plane early in your journey. You might choose to create a basic control plane and automate only some of the most commonly used functionality. You can progressively add more capabilities over time.

Design a control plane

After you've determined the requirements and the scope of your control plane, you need to design and architect it. A control plane is an important component. You need to plan it carefully, just as you would plan the other elements of your system.

Well-architected control planes

Because a control plane is its own system, it's important to consider all five pillars of the [Azure Well-Architected Framework](#) when you design one. The following sections highlight some particular areas to focus on.

Reliability

Control planes are often mission-critical components. It's essential that you plan the level of resiliency and reliability that your control plane needs.

Consider what happens if your control plane is unavailable. In extreme cases, a control plane outage might result in your entire solution being unavailable. Even if your control plane isn't a single point of failure, an outage might have some of the following effects:

- Your system isn't able to onboard new tenants, which might affect your sales and business growth.
- Your system can't manage existing tenants, which results in more calls to your support team.
- You can't measure the consumption of tenants or bill them for their usage, which results in lost revenue.
- You can't respond to a security incident by disabling or reconfiguring a tenant.
- Maintenance debt accumulates, which results in long-term damage to the system. For example, if your solution requires nightly cleanup of old data, your disks could

fill up or your performance could degrade.

Define [service-level objectives](#) for your control plane, including availability targets, the recovery time objective (RTO), and the recovery point objective (RPO). The objectives that you set for your control plane might be different from those that you offer your customers.

Follow the [Azure Well-Architected Framework guidance for building reliable solutions](#) throughout your system, including your control plane.

Security

Control planes are often highly privileged systems. Security problems within a control plane can have catastrophic consequences. Depending on its design and functionality, a control plane might be vulnerable to many different types of attacks, including the following:

- A control plane might have access to keys and secrets for all tenants. An attacker who has access to your control plane might be able to gain access to a tenant's data or resources.
- A control plane can often deploy new resources to Azure. Attackers might be able to exploit your control plane to deploy their own resources into your subscriptions, potentially incurring extensive charges.
- If an attacker successfully brings your control plane offline, there can be immediate and long-term damage to your system and to your business. See [Reliability](#) for example consequences of a control plane being unavailable.

When you design and implement a control plane, it's essential that you follow security best practices and create a comprehensive threat model to document and mitigate potential threats and security problems in your solution. For more information, see the [Azure Well-Architected Framework guidance for building secure solutions](#).

Operational excellence

Because a control plane is a critical component, you should carefully consider how you deploy and operate it in production.

Like other parts of your solution, you should deploy non-production instances of your control plane so that you can thoroughly test its functionality. If your control plane performs deployment operations, consider how your non-production control planes interact with your Azure environment, and which Azure subscription you deploy non-production resources to.

You should also plan how you govern your team's access to your control plane. Follow best practices for granting only the permissions that team members need to perform their duties. In addition to helping to avoid security incidents, this approach helps to reduce the effect of accidental misconfiguration.

Components

There's no single template for a control plane, and the components that you design and build depend on your requirements. Commonly, a control plane consists of APIs and background worker components. In some solutions, a control plane might also include a user interface.

Isolate your control plane from tenant workloads

It's a good practice to separate your control plane's resources from those used to serve your tenants' data planes. For example, you should consider using separate database servers, application servers and Azure App Service plans, and other components. It's often a good idea to keep your control plane's resources in a separate Azure resource group from those that contain tenant-specific resources.

By isolating your control plane from tenants' workloads, you gain several advantages:

- You can configure scaling separately. For example, your control plane might have consistent resource requirements, and your tenants' resources might scale elastically depending on their needs.
- There's a [bulkhead](#) between your control and data planes, which helps to prevent [noisy neighbor problems](#) from spreading between the planes of your solution.
- Control planes are typically highly privileged systems that have high levels of access. By separating the control plane from data planes, you reduce the likelihood that a security vulnerability might allow attackers to elevate their permissions across your entire system.
- You can deploy separate networking and firewall configurations. Data planes and control planes usually require different types of network access.

Orchestrate sequences of long-running operations

The operations that a control plane performs are often long-running and involve coordination between multiple systems. The operations can also have complex failure modes. When you design your control plane, it's important to use a suitable technology for coordinating long-running operations or workflows.

For example, suppose that, when you onboard a new tenant, your control plane runs the following actions in sequence:

1. **Deploy a new database.** This action is an Azure deployment operation. It might take several minutes to complete.
2. **Update your tenant metadata catalog.** This action might involve running a command against an Azure SQL database.
3. **Send a welcome email to the new tenant.** This action invokes a third-party API to send the email.
4. **Update your billing system to prepare to invoice the new tenant.** This action invokes a third-party API. You've noticed that it intermittently fails.
5. **Update your customer relationship management (CRM) system to track the new tenant.** This action invokes a third-party API.

If any step in the sequence fails, you need to consider what to do, such as:

- Retry the failed operation. For example, if your Azure SQL command in step 2 fails with a transient error, you could retry it.
- Continue to the next step. For example, you might decide that it's acceptable if the update to your billing system fails, because your sales team will manually add the customer.
- Abandon the workflow and trigger a manual recovery process.

You also need to consider what the user experience is like for each failure scenario.

Manage shared components

A control plane needs to be aware of any components that aren't dedicated to specific tenants, but instead are shared. Some components might be shared among all tenants within a stamp. Other components might be shared among all stamps in a region, or even shared globally across all regions and stamps. Whenever a tenant is onboarded, reconfigured, or offboarded, your control plane needs to know what to do with these shared components.

Some shared components might need to be reconfigured whenever a tenant is added or removed. For example, suppose you have a globally shared Azure Front Door profile. If you add a tenant with a custom domain name, your control plane might need to update the profile's configuration to route requests for that domain name to the correct application. Similarly, when a tenant is offboarded, your control plane might need to remove the custom domain name from the Azure Front Door profile to avoid [subdomain takeover attacks](#).

Shared components might have complex scaling rules that your control plane needs to follow. For example, suppose that you follow a [bin packing](#) approach to deploy your tenants' databases. When a new tenant is onboarded, you add a new Azure SQL database for that tenant, and then you assign it to an Azure SQL elastic pool. You might have determined that you need to increase the resources allocated to your pool for every tenth database that you add. When you add or remove a tenant, your control plane needs to re-evaluate the pool's configuration and decide whether to change the pool's resources. When you reach the maximum number of databases that you can assign to a single elastic pool, you need to create a new pool and start to use that pool for new tenant databases. Your control plane needs to take responsibility for managing each of these shared components, scaling and reconfiguring them whenever something changes.

When your control plane manages shared components, it's important to be aware of race conditions, which can occur when multiple operations happen in parallel. For example, if you onboard a new tenant at the same time that you offboard a different tenant, you need to ensure that your ultimate end state is consistent and meets your scaling requirements.

Use multiple control planes

In a complex environment, you might need to use multiple control planes, each with different areas of responsibility. Many multitenant solutions follow the [Deployment Stamps pattern](#) and shard tenants across multiple stamps. When you use this pattern, you might create separate control planes for global and stamp responsibilities.

Tip

Coordinating across multiple control planes is complex, so try to minimize the number of control planes that you build. Most solutions need only one control plane.

Global control planes

A global control plane is typically responsible for the overall management and tracking of tenants. A global control plane might have the following responsibilities:

- **Tenant placement.** The global control plane determines which stamp a tenant should use. It might make this determination based on factors like the tenant's

region, each stamp's capacity utilization, and the tenant's service level requirements.

- **Tenant onboarding and lifecycle management.** These responsibilities include tracking all tenants across all deployments.

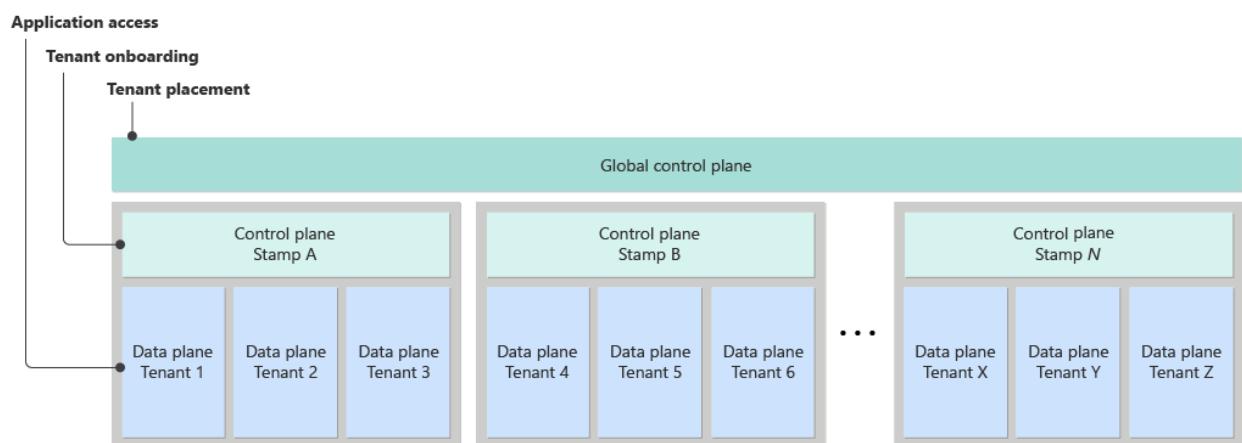
Stamp control planes

A stamp control plane is deployed into each deployment stamp and is responsible for the tenants and resources allocated to that stamp. A stamp control plane might have these responsibilities:

- **Creating and managing tenant-specific resources within the stamp**, like databases and storage containers
- **Managing shared resources**, including monitoring the consumption of shared resources and deploying new instances when they're approaching their maximum capacity
- **Performing maintenance operations within the stamp**, like database index management and cleanup operations

Each stamp's control plane coordinates with the global control plane. For example, suppose a new tenant signs up. The global control plane is initially responsible for selecting a stamp for the tenant's resources. Then, the global control plane prompts the stamp's control plane to create the necessary resources for the tenant.

The following diagram shows an example of how the two control planes might coexist in a single system:



Tenant control planes

Tenants might use a tenant-level control plane to manage their own logical or physical resources. A tenant control plane might have the following responsibilities:

- **Management of tenant-specific configuration**, like user access
- **Tenant-initiated maintenance operations**, like backing up data or downloading a previous backup
- **Update management**, if you allow tenants to [control their own updates to their applications](#)

The following diagram shows a complex system that has a global control plane, stamp control planes, and a control plane for each tenant:



Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Mick Alberts](#) | Technical Writer
- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure
- [Landon Pierce](#) | Customer Engineer, FastTrack for Azure
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirska](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Architectural considerations overview
- Azure Well-Architected Framework
- Architectural approaches for control planes in multitenant solutions

Measure the consumption of each tenant

Article • 12/14/2023

As a solution provider, it's important to measure the consumption of each tenant in your multitenant solution. By measuring the consumption of each tenant, you can ensure that the cost of goods sold (COGS), for delivering the service to each tenant, is profitable. On this page, we provide guidance for technical decision-makers about the importance of measuring consumption, and the approaches you can consider to measure a tenant's consumption, as well as the tradeoffs involved.

There are two primary concerns driving the need for measuring each tenant's consumption:

- You need to measure the actual cost to serve each tenant. This is important to monitor the profitability of the solution for each tenant.
- You need to determine the amount to charge the tenant, when you're using [consumption-based pricing](#).

However, it can be difficult to measure the actual resources used by a tenant in a multitenant solution. Most services that can be used as part of a multitenant solution don't automatically differentiate or break down usage, based on whatever you define a tenant to be. For example, consider a service that stores data for all of your tenants in a single relational database. It's difficult to determine exactly how much space each tenant uses of that relational database, either in terms of storage or of the compute capacity that's required to service any queries and requests.

By contrast, for a single-tenant solution, you can use [Azure Cost Management](#) within the Azure portal, to get a complete cost breakdown for all the Azure resources that are consumed by that tenant.

Therefore, when facing these challenges, it's important to consider how to measure consumption.

Note

In some cases, it's commercially acceptable to take a loss on delivering service to a tenant, for example, when you enter a new market or region. This is a commercial choice. Even in these situations, it's still a good idea to measure the consumption of each tenant, so that you can plan for the future.

Indicative consumption metrics

Modern applications (built for the cloud) are usually made up of many different services, each with different measures of consumption. For example, a storage account measures consumption based on the amount of data stored, the data transmitted, and the numbers of transactions. However, Azure App Service consumption is measured by the amount of compute resources allocated over time. If you have a solution that includes a storage account and App Service resources, then combining all these measurements together to calculate the actual COGS (cost of goods sold) can be a very difficult task. Often, it's easier to use a single indicative measurement to represent consumption in the solution.

For example, if a multitenant solution shares a single relational database, then the data stored by a tenant might be a good indicative consumption metric.

ⓘ Note

Even if you use the volume of data stored by a tenant as an indicative consumption measure, it might not be a true representation of consumption for every tenant. For example, if a particular tenant does a lot more reads or runs more reporting from the solution, but it doesn't write a lot of data, then it could use a lot more compute than the storage requirements would indicate.

It's important to occasionally measure and review the actual consumption across your tenants, to determine whether the assumptions you're making about your indicative metrics are correct.

Transaction metrics

An alternative way of measuring consumption is to identify a key transaction that is representative of the COGS for the solution. For example, in a document management solution, it could be the number of documents created. This can be useful, if there's a core function or feature within a system that is transactional, and if it can be easily measured.

This method is usually easy and cost effective to implement, as there's usually only a single point in your application that needs to record the number of transactions that occur.

Per-request metrics

In solutions that are primarily API-based, it might make sense to use a consumption metric that is based around the number of API requests being made to the solution. This can often be quite simple to implement, but it does require you to use APIs as the primary interface to the system. There will be an increased operational cost of implementing a per-request metric, especially for high volume services, because of the need to record the request utilization (for audit and billing purposes).

Note

User-facing solutions that consist of a single-page application (SPA), or a mobile application that uses the APIs, may not be a good fit for the per-request metric. This is because there's little understanding by the end user of the relationship between the use of the application and the consumption of APIs. Your application might be chatty (it makes many API requests) or chunky (it makes too few API requests), and the user wouldn't notice a difference.

Warning

Make sure you store request metrics in a reliable data store that's designed for this purpose. For example, although Azure Application Insights can track requests and can even track tenant IDs (by using **properties**), Application Insights is not designed to store every piece of telemetry. It removes data, as part of its **sampling behavior**. For billing and metering purposes, choose a data store that will give you a high level of accuracy.

Estimate consumption

When measuring the consumption for a tenant, it may be easier to provide an estimate of the consumption for the tenant, rather than trying to calculate the exact amount of consumption. For example, for a multitenant solution that serves many thousands of tenants in a single deployment, it's reasonable to approximate that the cost of serving the tenant is just a percentage of the cost of the Azure resources.

You might consider estimating the COGS for a tenant, in the following cases:

- You aren't using [consumption-based pricing](#).
- The usage patterns and cost for every tenant is similar, regardless of size.
- Each tenant consumes a low percentage (say, <2%), of the overall resources in the deployment.
- The per-tenant cost is low.

You might also choose to estimate consumption in combination with [indicative consumption metrics](#), [transaction metrics](#), or [per-request metrics](#). For example, for an application that primarily manages documents, the percentage of overall storage used by a tenant, to store its documents, gives a close enough representation of the actual COGS. This can be a useful approach, when measuring the COGS is difficult or when it would add too much complexity to the application.

On-charging your costs

In some solutions, you can charge your customers the COGS for their tenant's resources. For example, you might use [Azure resource tags](#) to allocate billable Azure resources to tenants. You can then determine the cost to each tenant for the set of resources that's dedicated to them, plus a margin for profit and operation.

ⓘ Note

Some [Azure services don't support tags](#). For these services, you'll need to attribute the costs to a tenant, based on the resource name, resource group, or subscription.

[Azure Cost Analysis](#) can be used to analyze Azure resource costs for single tenant solutions that use tags, resource groups, or subscriptions to attribute costs.

However, this becomes prohibitively complex in most modern multitenant solutions, because of the challenge of accurately determining the exact COGS to serve a single tenant. This method should only be considered for very simple solutions, solutions that have single-tenant resource deployments, or custom tenant-specific add-on features within a larger solution.

Some Azure services provide features that allow other methods of attribution of costs in a multitenant environment. For example, Azure Kubernetes Service supports [multiple node pools](#), where each tenant is allocated a node pool with [node pool tags](#), which are used to attribute costs.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Daniel Scott-Raynsford](#) | Partner Technology Strategist

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Chad Kittel](#) | Principal Software Engineer
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Consider the [update deployment model you'll use](#).

Considerations for updating a multitenant solution

Article • 03/02/2023

One of the benefits of cloud technology is continuous improvement and evolution. As a service provider, you need to apply updates to your solution: you might need to make changes to your Azure infrastructure, your code/applications, your database schemas, or any other component. It's important to plan how you update your environments. In a multitenant solution, it's particularly important to be clear about your update policy because some of your tenants might be reluctant to allow changes to their environments, or they might have requirements that limit the conditions under which you can update their service.

When planning a strategy to update your solution, you need to:

- Identify your tenants' requirements.
- Clarify your own requirements to operate your service.
- Find a balance that both you and your tenants can accept.
- Communicate your strategy clearly to your tenants and other stakeholders.

In this article, we provide guidance for technical decision-makers about the approaches you can consider to update your tenants' software, and the tradeoffs involved.

Your customers' requirements

Consider the following questions:

- **Expectations and requirements:** Do your customers have expectations or requirements about when they can be updated? These might be formally communicated to you in contracts or service-level agreements, or they might be informal.
- **Maintenance windows:** Do your customers expect service-defined or even self-defined maintenance windows? They might need to communicate to their own customers about any potential outages.
- **Regulations:** Do your customers have any regulatory concerns that require additional approval before updates can be applied? For example, if you provide a health solution that includes IoT components, you might need to get approval from the United States Food and Drug Administration (FDA) before applying an update.

- **Sensitivity:** Are any of your customers particularly sensitive or resistant to having updates applied? Try to understand why. For example, if they run a physical store or a retail website, they might want to avoid updates around Black Friday, because the risks are higher than potential benefits.
- **History:** What's your track record of successfully completing updates without any impact to your customers? You should follow good DevOps, testing, deployment, and monitoring practices to reduce the likelihood of outages, and to ensure that you quickly identify any issues that updates introduce. If your customers know that you're able to update their environments smoothly, they're less likely to object.
- **Rollback:** Will customers want to roll back updates if there's a breaking change?

Your requirements

You also need to consider the following questions from your own perspective:

- **Control you're willing to provide:** Is it reasonable for your customers to have control over when updates are applied? If you're building a solution used by large enterprise customers, the answer might be yes. However, if you're building a consumer-focused solution, it's unlikely you'll give any control over how you upgrade or operate your solution.
- **Versions:** How many versions of your solution can you reasonably maintain at one time? Remember that if you find a bug and need to hotfix it, you might need to apply the hotfix to all of the versions in use.
- **Consequences of old versions:** What's the impact of letting customers fall too far behind the current version? If you release new features on a regular basis, will old versions become obsolete quickly? Also, depending on your upgrade strategy and the types of changes, you might need to maintain separate infrastructures for each version of your solution. So, there might be both operational and financial costs, as you maintain support for older versions.
- **Rollback:** Can your deployment strategy support rollbacks to previous versions? Is this something you want to enable?

Note

Consider whether you need to take your solution offline for updates or maintenance. Generally, outage windows are seen as an outdated practice, and modern DevOps practices and cloud technologies enable you to avoid downtime during updates and maintenance. However, you need to design for zero-downtime deployments, so it's important to consider your update process when you plan your solution architecture.

Even if you don't plan for outages during your update process, you might still consider defining a regular maintenance window. A window can help to communicate to your customers that changes happen during specific times.

For more information on achieving zero-downtime deployments, see [Eliminate downtime through versioned service updates](#).

Find a balance

If you leave cadence of your service updates entirely to your tenants' discretion, they might choose to never update. It's important to allow yourself to update your solution, while factoring in any reasonable concerns or constraints that your customers might have. For example, if a customer is particularly sensitive to updates on a Friday because that's their busiest day of the week, then can you just as easily defer updates to Mondays, without impacting your solution?

One approach that can work well is to roll out updates on a tenant-by-tenant basis, using [one of the approaches described below](#). Give your customer notice of planned updates. Allow customers to temporarily opt out, but not forever; put a reasonable limit on when you will require the update to be applied.

Also, consider allowing yourself the ability to deploy security patches, or other critical hotfixes, with minimal or no advance notice.

Another approach can be to allow tenants to initiate their own updates, at a time of their choosing. Again, you should provide a deadline, at which point you apply the update on their behalf.

Warning

Be careful about enabling tenants to initiate their own updates. This is complex to implement, and it will require significant development and testing effort to deliver and maintain.

Whatever you do, ensure you have a process to monitor the health of your tenants, especially before and after updates are applied. Often, critical production incidents (also called *live-site incidents*) happen after updates to code or configuration. Therefore, it's important you proactively monitor for and respond to any issues to retain customer confidence. For more information about monitoring, see [Monitoring for DevOps](#).

Communicate with your customers

Clear communication is key to building your customers' confidence. It's important to explain the benefits of regular updates, including new features, bug fixes, resolving security vulnerabilities, and performance improvements. One of the benefits of a modern cloud-hosted solution is the ongoing delivery of features and updates.

Consider the following questions:

- Will you notify customers of upcoming updates?
- If you do, will you implicitly request permission by providing an opt-out process, and what are the limits on opting out?
- Do you have a scheduled maintenance window that you use when you apply updates?
- What if you have an emergency update, like a critical security patch? Can you force updates in those situations?
- If you can't proactively notify customer of upcoming updates, can you provide retrospective notifications? For example, can you update a page on your website with the list of updates that you've applied?
- How many separate versions of your system will you maintain in production?

Communicate with your customer support team

It's important that your own support team has full visibility into updates that have been applied to each tenant. Customer support representatives should be able to easily answer the following questions:

- Have updates recently been applied to a tenant's infrastructure?
- What was the nature of those updates?
- What was the previous version?
- How frequently are updates applied to this tenant?

If one of your customers has a problem because of an update, you need to ensure your customer support team has the information necessary to understand what's changed.

Deployment strategies to support updates

Consider how you will deploy updates to your infrastructure. This is heavily influenced by the [tenancy model](#) that you use. Three common approaches for deploying updates are deployment stamps, feature flags, and deployment rings. You can use these

approaches independently, or you can combine them together to meet more complex requirements.

In all cases, ensure that you have sufficient reporting and visibility, so that you know what version of infrastructure, software, or feature each tenant is on, what they are eligible to migrate to, and any time-related data associated with those states.

Deployment Stamps pattern

Many multitenant applications are a good fit for the [Deployment Stamps pattern](#), in which you deploy multiple copies of your application and other components.

Depending on your isolation requirements, you might deploy a stamp for each tenant, or shared stamps that run multiple tenants' workloads.

Stamps are a great way to provide isolation between tenants. They also provide you with flexibility for your update process, because you can roll out updates progressively across stamps, without affecting others.

Feature flags

[Feature flags](#) enable you to add functionality to your solution, while only exposing that functionality to a subset of your customers or tenants.

Consider using feature flags if either of these situations apply to you:

- You deploy updates regularly but want to avoid showing new functionality until it's fully implemented.
- You want to avoid applying changes in behavior until a customer opts in.

You can embed feature flag support into your application by writing code yourself, or by using a service like [Azure App Configuration](#).

Deployment rings

[Deployment rings](#) enable you to progressively roll out updates across a set of tenants or deployment stamps. You can assign a subset of tenants to each ring.

You can determine how many rings to create and what each ring means for your own solution. Commonly, organizations use the following rings:

- **Canary:** A canary ring includes your own test tenants and customers who want to have updates as soon as they are available, with the understanding that they may

receive more frequent updates, and that updates might not have been through as comprehensive a validation process as in the other things.

- **Early adopter:** An early adopter ring contains tenants who are slightly more risk-averse, but who are still prepared to receive regular updates.
- **Users:** Most of your tenants will belong to the *users* ring, which receives less frequent and more highly tested updates.

API versions

If your service exposes an external API, consider that any updates you apply might affect the way that customers or partners integrate with your platform. In particular, you need to be conscious of breaking changes to your APIs. Consider using [an API versioning strategy](#) to mitigate the risk of updates to your API.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Chad Kittel](#) | Principal Software Engineer
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

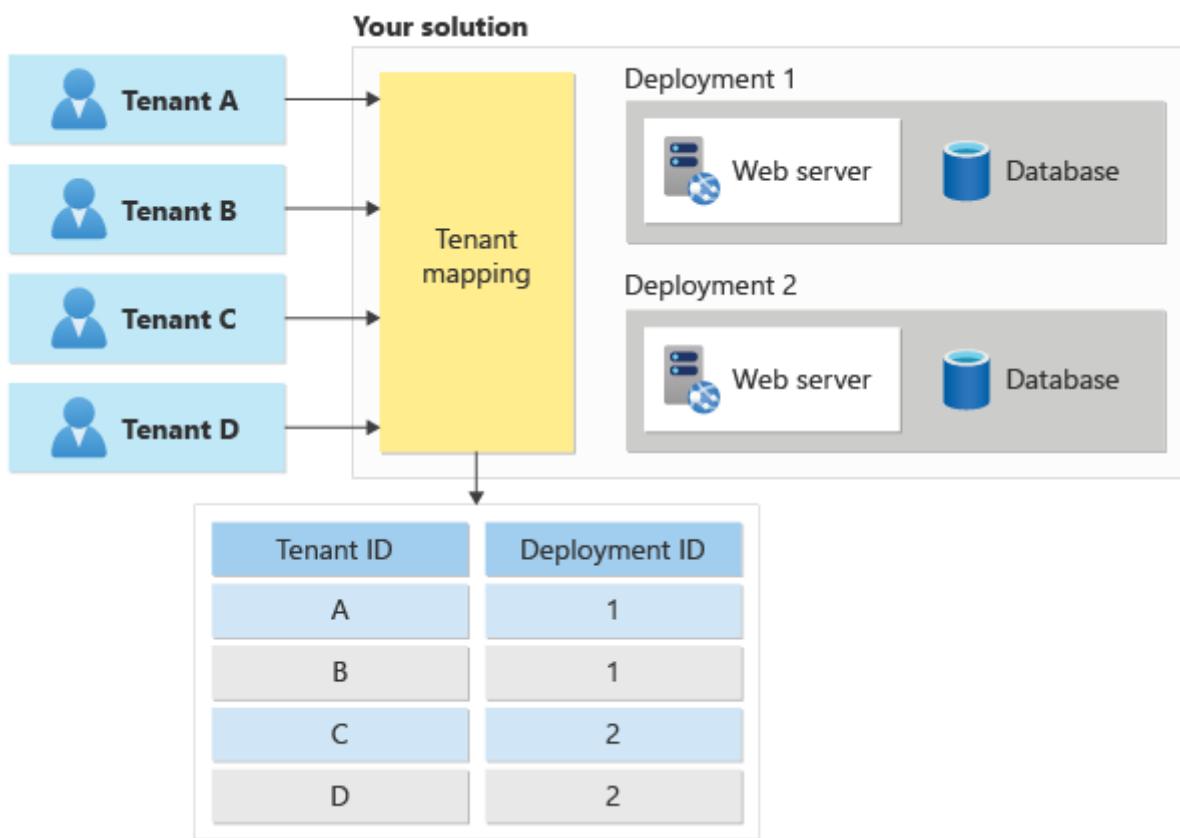
Next steps

- Consider when you would [map requests to tenants, in a multitenant solution](#).
- Review the [DevOps checklist](#) in Azure Well-Architected Framework.

Map requests to tenants in a multitenant solution

Azure

Whenever a request arrives into your application, you need to determine the tenant that the request is intended for. When you have tenant-specific infrastructure that may even be hosted in different geographic regions, you need to match the incoming request to a tenant. Then, you must forward the request to the physical infrastructure that hosts that tenant's resources, as illustrated below:



On this page, we provide guidance for technical decision-makers about the approaches you can consider to map requests to the appropriate tenant, and the tradeoffs involved in the approaches.

ⓘ Note

This page mostly discusses HTTP-based applications, like websites and APIs. However, many of same underlying principles apply to multitenant applications that use other communication protocols.

Approaches to identify tenants

There are multiple ways you can identify the tenant for an incoming request.

Domain names

If you use [tenant-specific domain or subdomain names](#), it's likely that requests can be easily mapped to tenants by using the `Host` header, or another HTTP header that includes the original hostname for each request.

However, consider the following questions:

- How will users know which domain name to use to access the service?
- Do you have a central entry point, like a landing page or login page, that all the tenants use? If you do, how will users identify the tenant that they need to access?
- What other information are you using to verify access to the tenant, such as authorization tokens? Do the authorization tokens include the tenant-specific domain names?

HTTP request properties

If you don't use tenant-specific domain names, you might still be able to use aspects of the HTTP request to identify the tenant that a particular request is for. For example, consider an HTTP request that identifies the tenant name as `tailwindtraders`. This might be communicated using the following:

- **The URL path structure**, such as `https://app.contoso.com/tailwindtraders/`.
- **A query string** in the URL, such as `https://contoso.com/app?tenant=tailwindtraders`.
- **A custom HTTP request header**, such as `X-Tenant-Id: tailwindtraders`.

Important

Custom HTTP request headers aren't useful where HTTP GET requests are issued from a web browser, or where the requests are handled by some types of web proxy. You should only use custom HTTP headers for GET operations when you're building an API, or if you control the client that issues the request and there's no web proxy included in the request processing chain.

When using this approach, you should consider the following questions:

- Will users know how to access the service? For example, if you use a query string to identify tenants, will a central landing page need to direct users to the correct tenant, by adding the query string?
- Do you have a central entry point, like a landing page or login page, that all tenants use? If you do, how will users identify the tenant that they need to access?
- Does your application provide APIs? For example, is your web application a single-page application (SPA) or a mobile application with an API backend? If it is, you might be able to use an [API gateway](#) or [reverse proxy](#) to perform tenant mapping.

Token claims

Many applications use claims-based authentication and authorization protocols, such as OAuth 2.0 or SAML. These protocols provide authorization tokens to the client. A token contains a set of *claims*, which are pieces of information about the client application or user. Claims can be used to communicate information like a user's email address. Your system can then include the user's email address, look up the user-to-tenant mapping, and then forward the request to the appropriate deployment. Or, you might even include the tenant mapping in your identity system, and add a tenant ID claim to the token.

If you are using claims to map requests to tenants, you should consider the following questions:

- Will you use a claim to identify a tenant? Which claim will you use?
- Can a user be a member of multiple tenants? If this is possible, then how will users select the tenants they'd like to work with?
- Is there a central authentication and authorization system for all tenants? If not, how will you ensure that all token authorities issue consistent tokens and claims?

API keys

Many applications expose APIs. These might be for internal use within your organization, or for external use by partners or customers. A common method of authentication for APIs is to use an *API key*. API keys are provided with each request, and they can be used to look up the tenant.

API keys can be generated in several ways. A common approach is to generate a cryptographically random value and store it in a lookup table, alongside the tenant ID. When a request is received, your system finds the API key in the lookup table, and it then matches it to a tenant ID. Another approach is to create a meaningful string with a tenant ID included inside it, and then you would digitally sign the key, by using an

approach like [HMAC](#). When you process each request, you verify the signature and then extract the tenant ID.

ⓘ Note

API keys don't provide a high level of security because they need to be manually created and managed, and because they don't contain claims. A more modern and secure approach is to use a claims-based authorization mechanism with short-lived tokens, such as OAuth 2.0 or OpenID Connect.

Consider the following questions:

- How will you generate and issue API keys?
- How will your API clients securely receive and store the API key?
- Do you need your API keys to expire after a period of time? How will you rotate your clients' API keys, without causing downtime?
- Does just relying on customer-rolled API keys provide an adequate level of security for your APIs?

ⓘ Note

API keys are not the same as passwords. API keys must be generated by the system, and they must be unique across all the tenants, so that each API key can be uniquely mapped to a single tenant. API gateways, such as [Azure API Management](#), can generate and manage API keys, validate keys on incoming requests, and map keys to individual API subscribers.

Client certificates

Client certificate authentication, sometimes called mutual TLS (mTLS), is commonly used for service-to-service communication. Client certificates provide a secure way to authenticate clients. Similarly to tokens and claims, client certificates provide *attributes* that can be used to determine the tenant. For example, the *subject* of the certificate may contain the email address of the user, which can be used to look up the tenant.

When planning to use client certificates for tenant mapping consider the following:

- How will you safely issue and renew the client certificates that are trusted by your service? Client certificates can be complex to work with, since they require special infrastructure to manage and issue certificates.

- Will client certificates be used only for initial login requests, or attached to all requests to your service?
- Will the process of issuing and managing certificates become unmanageable when you have a large number of clients?
- How will you implement the mapping between the client certificate and the tenant?

Reverse proxies

A reverse proxy, also referred to as an application proxy, can be used to route HTTP requests. A reverse proxy accepts a request from an ingress endpoint, and it can forward the request to one of many backend endpoints. Reverse proxies are useful for multitenant applications since they can perform the mapping between some piece of request information, offloading the task from your application infrastructure.

Many reverse proxies can use the properties of a request to make a decision about tenant routing. They can inspect the destination domain name, URL path, query string, HTTP headers, and even claims within tokens.

The following common reverse proxies are used in Azure:

- [Azure Front Door](#) is a global load balancer and web application firewall. It uses the Microsoft global edge network to create fast, secure, and highly scalable web applications.
- [Azure Application Gateway](#) is a managed web traffic load balancer that you deploy into the same physical region as your backend service.
- [Azure API Management](#) is optimized for API traffic.
- Commercial and open-source technologies (that you host yourself) include nginx, Traefik, and HAProxy.

Request validation

It is important that your application validates that any requests that it receives are authorized for the tenant. For example, if your application uses a custom domain name to map requests to the tenant, then your application must still check that each request received by the application is authorized for that tenant. Even though the request includes a domain name or other tenant identifier, it doesn't mean you should automatically grant access. When you use OAuth 2.0, you perform the validation by inspecting the *audience* and *scope* claims.

 Note

This is part of the *assume zero trust* security design principle in the [Microsoft Azure Well-Architected Framework](#).

When implementing request validation, you should consider the following:

- How will you authorize all the requests to your application? You need to authorize requests, regardless of the approach you use to map them to physical infrastructure.
- Use trusted, widely used and well maintained authentication and authorization frameworks and middleware, instead of implementing all of the validation logic yourself. For example, don't build token signature validation logic or client certificate cryptography libraries. Instead, use features of your application platform (or known trusted packages) that have been validated and tested.

Performance

Tenant mapping logic likely runs on every request to your application. Consider how well the tenant mapping process will scale, as your solution grows. For example, if you query a database table as part of your tenant mapping, will the database support a large amount of load? If your tenant mapping requires decrypting a token, will the computation requirements become too high over time? If your traffic is fairly modest, then this isn't likely to affect your overall performance. When you have a high-scale application, though, the overhead involved in this mapping can become significant.

Session affinity

One approach to reducing the performance overhead of tenant mapping logic is to use *session affinity*. Rather than perform the mapping on every request, consider computing the information only on the first request for each session. Your application then provides a *session cookie* to the client. The client passes the session cookie back to your service with all subsequent client requests within that session.

ⓘ Note

Many networking and application services in Azure can issue session cookies and natively route requests by using session affinity.

Consider the following questions:

- Can you use session affinity to reduce the overhead of mapping requests to tenants?
- What services do you use to route requests to the physical deployments for each tenant? Do these services support cookie-based session affinity?

Tenant migration

Tenants often need to be moved to new infrastructure as part of the [tenant lifecycle](#). When a tenant is moved to a new deployment, the HTTP endpoints they access might change. When this happens, consider that your tenant-mapping process needs to be updated. You may need to consider the following:

- If your application uses domain names for mapping requests, then it might also require a DNS change at the time of the migration. The DNS change might take time to propagate to clients, depending on the time-to-live of the DNS entries in your DNS service.
- If your migration changes the addresses of any endpoints during the migration process, then consider temporarily redirecting requests for the tenant to a maintenance page that automatically refreshes.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Daniel Scott-Raynsford](#) | Partner Technology Strategist

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Learn about [considerations when you work with domain names in a multitenant application](#).

Architectural considerations for identity in a multitenant solution

Article • 11/01/2023

Identity is an important aspect of any multitenant solution. The identity components of your application are responsible for both of the following:

- Verifying who a user is (*authentication*).
- Enforcing the user's permissions within the scope of a tenant (*authorization*).

Your customers might also wish to authorize external applications to access their data or to integrate to your solution. A user's identity determines what information a user or service will get access to. It's important that you consider your identity requirements, in order to isolate your application and data between tenants.

⊗ Caution

Authentication and authorization services, within multitenant and SaaS applications, are usually provided by a third-party identity provider (IdP). An identity provider is usually an integral part of an identity as a service (IDaaS) platform.

Building your own IdP is complex, expensive, and difficult to build securely. Building your own identity provider is **an antipattern**. We don't recommend it.

Before defining a multitenant identity strategy, you should first consider the high-level identity requirements of your service, including the following requirements:

- Will a user or [workload identities](#) be used to access a single application or multiple applications within a product family? For example, a retail product family might have both, a point-of-sale application and an inventory management application, which share the same identity solution.
- Are you planning on implementing modern authentication and authorization, such as OAuth2 and OpenID Connect?
- Does your solution only provide authentication to your UI-based applications? Or, do you also provide API access to your tenants and third parties?
- Will tenants need to federate to their own IdP, and will multiple different identity providers need to be supported for each tenant? For example, you might have tenants with Microsoft Entra ID, Auth0, and Active Directory Federation Services (ADFS), where each wishes to federate with your solution. You also need to

understand which federation protocols of your tenants' IdPs you'll support, because the protocols influence the requirements for your own IdP.

- Are there specific compliance requirements that they need to meet, such as [GDPR](#)?
- Do your tenants require their identity information to be located within a specific geographic region?
- Do users of your solution require access to data from one tenant or from multiple tenants within your application? Do they need the ability to quickly switch between tenants or to view consolidated information from multiple tenants? For example, users who have signed into the Azure portal can easily switch between different Microsoft Entra ID directories and subscriptions that they have access to.

When you've established your high-level requirements, you can start to plan more specific details and requirements, such as user directory sources and sign-up/sign-in flows.

Identity directory

For a multitenant solution to authenticate and authorize a user or service, it needs a place to store identity information. A *directory* can include authoritative records for each identity, or it might contain references to external identities that are stored in another identity provider's directory.

When you design an identity system for your multitenant solution, you need to consider which of the following types of IdP that your tenants and customers might need:

- **Local identity provider.** A local identity provider allows users to sign themselves up to the service. Users provide a username, an email address, or an identifier, such as a rewards card number. They also provide a credential, like a password. The IdP stores both the user identifier and the credentials.
- **Social identity provider.** A social identity provider allows users to use an identity that they have on a social network or other public identity provider, such as Facebook, Google, or a personal Microsoft account.
- **Use the tenant's Microsoft Entra ID directory.** Tenants might already have their own Microsoft Entra ID directory, and they might want your solution to use their directory as the IdP for accessing your solution. This approach is possible when your solution is built as [a multitenant Microsoft Entra application](#).
- **Federation with a tenant's identity provider.** Tenants might have their own IdP, other than Microsoft Entra ID, and they might want your solution to federate with it. Federation enables single sign-on (SSO) experiences, and it enables tenants to manage the lifecycle and security policies of their users, independently of your solution.

You should consider if your tenants need to support multiple identity providers. For example, you might need to support local identities, social identities, and federated identities, all within a single tenant. This requirement is common when companies use a solution that's both for their own employees and for contractors. They might use federation for their own employees' access to the solution, while also allowing access to contractors or to guest users, who don't have an account in the federated IdP.

Store authentication and tenant authorization information

In a multitenant solution, you need to consider where to store several types of identity information, including the following types:

- Details about user and service accounts, including their names and other information that's required by your tenants.
- Information that's required to securely authenticate your users, including information that's required to provide multifactor authentication (MFA).
- Tenant-specific information, such as tenant roles and permissions. This information is used for authorization.

⊗ Caution

We don't recommend building authentication processes yourself. Modern IdPs provide these authentication services to your application, and they also include other important features, such as MFA and conditional access. **Building your own identity provider is an antipattern**. We don't recommend it.

Consider the following options for storing identity information:

- Store all identity and authorization information in the IdP directory, and share it between multiple tenants.
- Store the user credentials in the IdP directory, and store the authorization information in the application tier, alongside the tenant information.

Determine the number of identities for a user

It's common for multitenant solutions to allow a user or workload identity to access the application and data of multiple tenants. Consider whether this approach is required for your solution. If it is, then you should consider the following questions:

- Should you create a single user identity for each person, or create separate identities for each tenant-user combination?
 - It's best to use a single identity for each person, wherever possible. It becomes difficult to manage multiple user accounts, both for you, as the solution vendor, and also for your users. Additionally, many of the intelligent threat protections offered by a modern IdP rely on each person having a single user account.
 - However, in some situations, it might be necessary for a user to have multiple distinct identities. For example, if people use your system both for work and personal purposes, they might want to separate their user accounts. Or, if your tenants have strict regulatory or geographical data storage requirements, they might require a person to have multiple identities so that they can comply with regulations or laws.
- If you use per-tenant identities, avoid storing credentials multiple times. Users should have their credentials stored against a single identity, and you should use features like guest identities to refer to the same user credentials from multiple tenants' identity records.

Grant users access to tenant data

Consider how users will be mapped to a tenant. For example, during the sign-up process, you might use a unique invitation code that they enter the first time they access a tenant. In some solutions, you might use the domain name of the users' sign-up email addresses as a way to identify the tenant that they belong to. Or, you might use another attribute of the user's identity record to map the user to a tenant. You should then store the mapping based on the underlying immutable unique identifiers for both the tenant and the user.

If your solution is designed so that a single user is only ever going to access the data for a single tenant, then consider the following decisions:

- How does the IdP detect which tenant a user is accessing?
 - How does the IdP communicate the tenant identifier to the application?
- Commonly, a tenant identifier claim is added to the token.

If a single user needs to be granted access to multiple tenants, then you need to consider the following decisions:

- How does your solution identify and grant permissions to a user who has access to multiple tenants? For example, could a user be an administrator in a training tenant, and have read-only access to a production tenant? Or, could you have separate tenants for different departments in an organization, but you need to maintain consistent user identities across all of the tenants?

- How does a user switch between tenants?
- If you use workload identities, how does a workload identity specify the tenant that it needs to access?
- Is there tenant-specific information stored in the user's identity record that could leak information between tenants? For example, suppose a user signed up with a social identity and was then granted access to two tenants. Tenant A enriched the user's identity with more information. Should tenant B have access to the enriched information?

User sign-up process for local identities or social identities

Some tenants might need to allow users to sign themselves up for an identity in your solution. Self-service sign-up might be required if you don't require federation with a tenant's identity provider. If a self-sign up process is needed, then you should consider the following questions:

- Which identity sources are users allowed to sign up from? For example, can a user create a local identity and also use a social identity provider?
- If only local identities are allowed, will only specific email domains be allowed? For example, can a tenant specify that only users who have an @contoso.com email address are permitted to sign up?
- What is the user principal name (UPN) that should be used to uniquely identify each local identity during the sign-in process? For example, an email address, username, phone number, and rewards card number are all common choices for UPNs. However, UPNs can change over time. When you refer to the identity in your application's authorization rules or audit logs, it's a good practice to use the underlying immutable unique identifier of the identity. Microsoft Entra ID provides an object ID (OID), which is an immutable identifier.
- Will a user be required to verify their UPN? For example, if the user's email address or phone number is used as a UPN, how will you verify the information is accurate?
- Do tenant administrators need to approve sign-ups?
- Do tenants require a tenant-specific sign-up experience or URL? For example, do your tenants require a branded sign-up experience when users sign up? Do your tenants require the ability to intercept a sign-up request and perform extra validation before it proceeds?

Tenant access for self sign-up users

When users are allowed to sign themselves up for an identity, there usually needs to be a process for them to be granted access to a tenant. The sign-up flow might include an access grant process, or it could be automated, based on the information about the users, such as their email addresses. It's important to plan for this process and consider the following questions:

- How will the sign-up flow determine that a user should be granted access to a specific tenant?
- If users should no longer have access to a tenant, will your solution automatically revoke their access? For example, when users leave an organization, there needs to be a manual or automated process that removes their access from the tenant.
- Do you need to provide a way for tenants to audit the users who have access to their tenants and their permissions?

Automated account lifecycle management

A common requirement for corporate or enterprise customers of a solution is a set of features that allows them to automate account onboarding and off-boarding. Open protocols, such as [System for Cross-domain Identity Management \(SCIM\)](#), provide an industry-standard approach to automation. This automated process usually includes not only creation and removal of identity records, but also management of tenant permissions. Consider the following questions when you implement automated account lifecycle management in a multitenant solution:

- Do your customers need to configure and manage an automated lifecycle process per tenant? For example, when a user is onboarded, you might need to create the identity within multiple tenants in your application, where each tenant has a different set of permissions.
- Do you need to implement SCIM, or can you provide tenants federation instead, to keep the source of truth for users under the control of the tenant, instead of managing local users?

User authentication process

When a user signs into a multitenant application, your identity system authenticates the user. You should consider the following questions, when you plan your authentication process:

- Do your tenants need to configure their own multifactor authentication (MFA) policies? For example, if your tenants are in the financial services industry, they

need to implement strict MFA policies, while a small online retailer might not have the same requirements.

- Do your tenants need to configure their own conditional access rules? For example, different tenants might need to block sign-in attempts from specific geographic regions.
- Do your tenants need to customize the sign-in process for each tenant? For example, do you need to show a customer's logo? Or, does information about each user need to be extracted from another system, such as a rewards number, and returned to the identity provider to add to the user profile?
- Do your users need to impersonate other users? For example, a support team member might wish to investigate an issue that another user has, by impersonating their user account without having to authenticate as the user.
- Do your users need to gain access to the APIs for your solution? For example, users or third-party applications might need to directly call your APIs to extend your solution, without a user interface to provide an authentication flow.

Workload identities

In most solutions, an identity often represents a user. Some multitenant systems also allow *workload identities* to be used by *services* and *applications*, to gain access to your application resources. For example, your tenants might need to access an API that's provided by your solution, so that they can automate some of their management tasks.

Workload identities are similar to user identities, but usually they require different authentication methods, such as keys or certificates. Workload identities don't use MFA. Instead, workload identities usually require other security controls, such as regular key-rolling and certificate expiration.

If your tenants expect to be able to enable workload identity access to your multitenant solution, then you should consider the following questions:

- How will workload identities be created and managed in each tenant?
- How will workload identity requests be scoped to the tenant?
- Do you need to limit the number of workload identities that are created by each tenant?
- Do you need to provide conditional access controls on workload identities for each tenant? For example, a tenant might want to limit a workload identity from being authenticated from outside a specific region.
- Which security controls will you provide to tenants to ensure that workload identities are kept secure? For example, automated key rolling, key expiration,

certificate expiration, and sign-in risk monitoring are all methods of reducing the risk, where a workload identity might be misused.

Federate with an identity provider for single sign-on (SSO)

Tenants, who already have their own user directories, might want your solution to *federate* to their directories. Federation allows your solution to use the identities in their directory, instead of managing another directory with distinct identities.

Federation is particularly important when some tenants would like to specify their own identity policies, or to enable single sign-on (SSO) experiences.

If you're expecting tenants to federate with your solution, you should consider the following questions:

- What is the process for configuring the federation for a tenant? Can tenants configure federation themselves, or does the process require manual configuration and maintenance by your team?
- Which federation protocols will you support?
- What processes are in place to ensure federation can't be misconfigured, to grant access to another tenant?
- Will a single tenant's identity provider need to be federated to more than one tenant in your solution? For example, if customers have both a training and production tenant, they might need to federate the same identity provider to both tenants.

Authorization models

Decide on the authorization model that makes the most sense for your solution. Two common authorization approaches are:

- **Role-based authorization.** Users are assigned to roles. Some features of the application are restricted to specific roles. For example, a user in the administrator role can perform any action, while a user in a lower role might have a subset of permissions throughout the system.
- **Resource-based authorization.** Your solution provides a set of distinct resources, each of which has its own set of permissions. A specific user might be an administrator of one resource and have no access to another resource.

These models are distinct, and the approach you select affects your implementation and the complexity of the authorization policies that you can implement.

For more information, see [Role-based and resource-based authorization](#).

Entitlements and licensing

In some solutions, you might use [per-user licensing](#) as part of your commercial pricing model. You would provide different tiers of user licenses with different capabilities. For example, users with one license might be permitted to use a subset of the features of the application. The capabilities that specific users are allowed to access, based on their licenses, is sometimes called an *entitlement*.

Although tracking and enforcing entitlements is similar to authorization, it's ordinarily handled by the application code or by a dedicated entitlements system, rather than managed by the identity system.

Identity scale and authentication volume

As multitenant solutions grow, the number of users and sign-in requests that need to be processed by the solution will increase. You should consider the following questions:

- Will the user directory scale to the number of users that are required?
- Will the authentication process handle the expected number of sign-ins and sign-ups?
- Will there be spikes that the authentication system can't handle? For example, at 9am PST, everyone in the western United States region might start work and sign in to your solution, causing a spike in sign-in requests. These situations are sometimes called *login storms*.
- Can high load in other parts of your solution impact the performance of the authentication process? For example, if your authentication process requires calling into an application-tier API, will high numbers of authentication requests cause problems for the rest of your solution?
- What will happen if your IdP becomes unavailable? Is there a backup authentication service that can take over to provide business continuity, while the IdP is unavailable?

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Jelle Druyts](#) | Principal Customer Engineer, FastTrack for Azure
- [Sander van den Hoven](#) | Senior Partner Technology Strategist
- [Nick Ward](#) | Senior Cloud Solution Architect

Next steps

Review [Architectural approaches for identity in multitenant solutions](#).

Considerations when using domain names in a multitenant solution

Azure

In many multitenant web applications, a domain name can be used as a way to identify a tenant, to help with routing requests to the correct infrastructure, and to provide a branded experience to your customers. Two common approaches are to use subdomains and custom domain names. On this page, we provide guidance for technical decision-makers about the approaches you can consider and their tradeoffs.

Subdomains

Each tenant might get a unique subdomain under a common shared domain name, using a format like `tenant.provider.com`.

Let's consider an example multitenant solution built by Contoso. Customers purchase Contoso's product to help manage their invoice generation. All of Contoso's tenants might be assigned their own subdomain, under the `contoso.com` domain name. Or, if Contoso uses regional deployments, they might assign subdomains under the `us.contoso.com` and `eu.contoso.com` domains. In this article, we refer to these as *stem domains*. Each customer gets their own subdomain under your stem domain. For example, Tailwind Toys might be assigned `tailwind.contoso.com`, and Adventure Works might be assigned `adventureworks.contoso.com`.

ⓘ Note

Many Azure services use this approach. For example, when you create an Azure storage account, it is assigned a set of subdomains for you to use, such as `<your account name>.blob.core.windows.net`.

Manage your domain namespace

When you create subdomains under your own domain name, you need to be mindful that you could have multiple customers with similar names. Because they share a single

stem domain, the first customer to get a particular domain will get their preferred name. Then, subsequent customers have to use alternate subdomain names, because full domain names must be globally unique.

Wildcard DNS

Consider using wildcard DNS entries to simplify the management of subdomains. Instead of creating DNS entries for `tailwind.contoso.com`, `adventureworks.contoso.com`, and so forth, you could instead create a wildcard entry for `*.contoso.com` and direct all subdomains to single IP address (A record) or canonical name (CNAME record).

 **Note**

Make sure that your web-tier services support wildcard DNS, if you plan to rely on this feature. Many Azure services, including Azure Front Door and Azure App Service, support wildcard DNS entries.

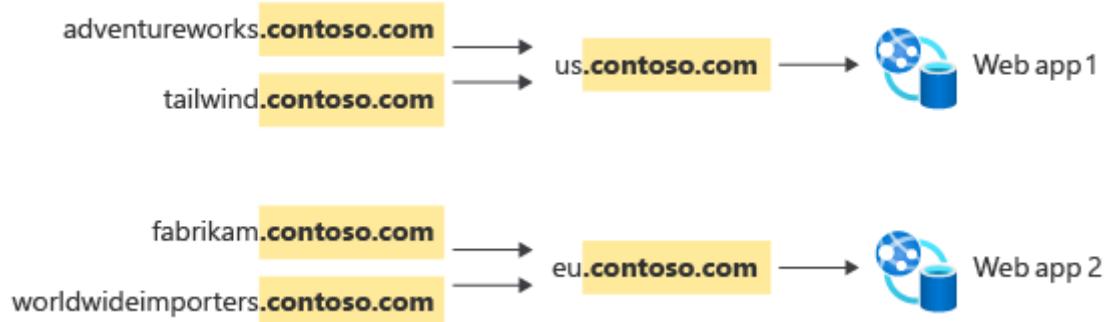
Subdomains with multipart stem domains

Many multitenant solutions are spread across multiple physical deployments. This is a common approach when you need to comply with data residency requirements, or when you want to provide better performance by deploying resources geographically closer to the users.

Even within a single region, you might also need to spread your tenants across independent deployments, to support your scaling strategy. If you plan to use subdomains for each tenant, you might consider a multipart subdomain structure.

Here's an example: Contoso publishes a multitenant application for its four customers. Adventure Works and Tailwind Traders are in the United States, and their data is stored on a shared US instance of the Contoso platform. Fabrikam and Worldwide Importers are in Europe, and their data is stored on a European instance.

If Contoso chose to use a single stem domain, `contoso.com`, for all their customers, here's what this might look like:



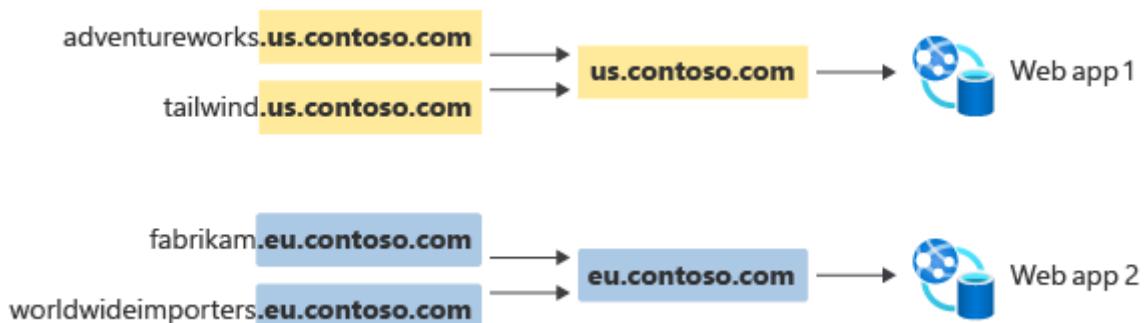
The DNS entries (that are required to support this configuration) might look like this:

[\[\] Expand table](#)

Subdomain	CNAME to
adventureworks.contoso.com	us.contoso.com
tailwind.contoso.com	us.contoso.com
fabrikam.contoso.com	eu.contoso.com
worldwideimporters.contoso.com	eu.contoso.com

Each new customer that is onboarded requires a new subdomain, and the number of subdomains grows with each customer.

Alternatively, Contoso could use deployment- or region-specific stem domains, like this:



Then, by using wildcard DNS, the DNS entries for this deployment might look like this:

[\[\] Expand table](#)

Subdomain	CNAME to
*.us.contoso.com	us.contoso.com
*.eu.contoso.com	eu.contoso.com

Contoso doesn't need to create subdomain records for every customer. Instead, they have a single wildcard DNS record for each geography's deployment, and any new customers who are added underneath that stem automatically inherit the CNAME record.

There are benefits and drawbacks to each approach. When using a single stem domain, each tenant you onboard requires a new DNS record to be created, which introduces more operational overhead. However, you have more flexibility to move tenants between deployments, because you can change the CNAME record to direct their traffic to another deployment. This change won't affect any other tenants. When using multiple stem domains, there's a lower management overhead. Also, you can reuse customer names across multiple regional stem domains, because each stem domain effectively represents its own namespace.

Custom domain names

You might want to enable your customers to bring their own domain names. Some customers see this as an important aspect of their branding. Custom domain names might also be required to meet customers' security requirements, especially if they need to supply their own TLS certificates. While it might seem trivial to enable customers to bring their own domain names, there are some hidden complexities to this approach, and it requires thoughtful consideration.

Name resolution

Ultimately, each domain name needs to be resolved to an IP address. As you've seen, the approach by which name resolution happens can depend on whether you deploy a single instance or multiple instances of your solution.

Let's return to our example. One of Contoso's customers, Fabrikam, has asked to use `invoices.fabrikam.com`, as their custom domain name to access Contoso's service. Because Contoso has multiple deployments of their platform, they decide to use subdomains and CNAME records to achieve their routing requirements. Contoso and Fabrikam configure the following DNS records:

Name	Record type	Value	Configured by
invoices.fabrikam.com	CNAME	fabrikam.eu.contoso.com	Fabrikam
*.eu.contoso.com	CNAME	eu.contoso.com	Contoso
eu.contoso.com	A	(Contoso's IP address)	Contoso

From a name resolution perspective, this chain of records accurately resolves requests for `invoices.fabrikam.com` to the IP address of Contoso's European deployment.

Host header resolution

Name resolution is only half of the problem. All of the web components within Contoso's European deployment need to be aware of how to handle requests that arrive with Fabrikam's domain name in their `Host` request header. Depending on the specific web technologies that Contoso uses, this might require further configuration for each tenant's domain name, which adds extra operational overhead to the onboarding of tenants.

You can also consider rewriting host headers, so that regardless of the incoming request's `Host` header, your web server sees a consistent header value. For example, Azure Front Door enables you to rewrite `Host` headers, so that regardless of the request, your application server receives a single `Host` header. Azure Front Door propagates the original host header in the `X-Forwarded-Host` header, so that your application can inspect it and then look up the tenant. However, rewriting a `Host` header can cause other problems. For more information, see [Host name preservation](#).

Domain validation

It's important to validate the ownership of custom domains before onboarding them. Otherwise, you risk a customer accidentally or maliciously *parking* a domain name.

Let's consider Contoso's onboarding process for Adventure Works, who have asked to use `invoices.adventureworks.com` as their custom domain name. Unfortunately, somebody made a typo when they tried to onboard the custom domain name, and they

missed the *s*. So, they set it up as `invoices.adventurework.com`. Not only does the traffic not flow correctly for Adventure Works, but when another company named *Adventure Work* tries to add their custom domain to Contoso's platform, they're told the domain name is already in use.

When working with custom domains, especially within a self-service or automated process, it's common to require a domain verification step. This might require that the CNAME records be set up before the domain can be added. Alternatively, Contoso might generate a random string and ask Adventure Works to add a DNS TXT record with the string value. That would prevent the domain name from being added, until the verification is completed.

Dangling DNS and subdomain takeover attacks

When you work with custom domain names, you are potentially vulnerable to a class of attack called *dangling DNS or subdomain takeover*. This attack happens when customers disassociate their custom domain name from your service, but they don't delete the record from their DNS server. This DNS entry then points to a non-existent resource and is vulnerable to a takeover.

Let's consider how Fabrikam's relationship with Contoso might change:

1. Fabrikam has decided to no longer work with Contoso, and so they have terminated their business relationship.
2. Contoso has offboarded the Fabrikam tenant, and they requested for `fabrikam.contoso.com` to no longer work. However, Fabrikam forgot to delete the CNAME record for `invoices.fabrikam.com`.
3. A malicious actor creates a new Contoso account and gives it the name `fabrikam`.
4. The attacker onboards the custom domain name `invoices.fabrikam.com` to their new tenant. Since Contoso performs CNAME-based domain validation, they check Fabrikam's DNS server. They see that the DNS server returns a CNAME record for `invoices.fabrikam.com`, which points to `fabrikam.contoso.com`. Contoso considers the custom domain validation to be successful.
5. If any Fabrikam employees tried to access the site, requests would appear to work. If the attacker sets up their Contoso tenant with Fabrikam's branding, employees might be fooled into accessing the site and providing sensitive data, which the attacker can then access.

Common strategies to protect against dangling DNS attacks are:

- Require that the CNAME record is deleted *before* the domain name can be removed from the tenant's account.

- Prohibit the reuse of tenant identifiers, and also require that the tenant create a TXT record with a name matching the domain name and a randomly generated value, which changes for each onboarding attempt.

TLS/SSL certificates

Transport Layer Security (TLS) is an essential component when working with modern applications. It provides trust and security to your web applications. The ownership and management of TLS certificates is something that needs careful consideration for multitenant applications.

Typically, the owner of a domain name will be responsible for issuing and renewing its certificates. For example, Contoso is responsible for issuing and renewing TLS certificates for `us.contoso.com`, as well as a wildcard certificate for `*.contoso.com`.

Similarly, Fabrikam would generally be responsible for managing any records for the `fabrikam.com` domain, including `invoices.fabrikam.com`. The CAA (Certificate Authority Authorization) DNS record type can be used by a domain owner. CAA records ensure that only specific authorities can create certificates for the domain.

If you plan to allow customers to bring their own domains, consider whether you plan to issue the certificate on the customer's behalf, or whether the customers must bring their own certificates. Each option has benefits and drawbacks.

- If you issue a certificate for a customer, you can handle the renewal of the certificate, so the customer doesn't have to remember to keep it updated. However, if the customers have CAA records on their domain names, they might need to authorize you to issue certificates on their behalf.
- If you expect customers should issue and provide you with their own certificates, you are responsible for receiving and managing the private keys in a secure manner, and you might have to remind your customers to renew the certificate before it expires, to avoid an interruption in their service.

Several Azure services support automatic management of certificates for custom domains. For example, Azure Front Door and App Service provide certificates for custom domains, and they automatically handle the renewal process. This removes the burden of managing certificates, from your operations team. However, you still need to consider the question of ownership and authority, such as whether CAA records are in effect and configured correctly. Also, you need to ensure your customers' domains are configured to allow the certificates that are managed by the platform.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Tip

Many services use Azure Front Door to manage domain names. For information about how to use Azure Front Door in a multitenant solution, see [Use Azure Front Door in a multitenant solution](#).

Return to the [architectural considerations overview](#). Or, review the [Microsoft Azure Well-Architected Framework](#).

Architectural approaches for a multitenant solution

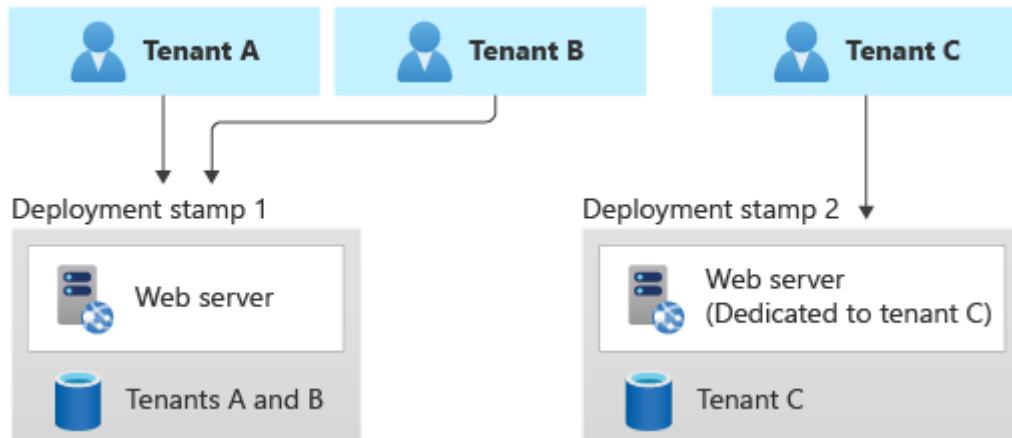
Azure

There are many different ways that you can design and build multitenant solutions in Azure. At one extreme, you can share every resource in your solution among all of your tenants. At the other extreme, you can deploy isolated resources for every tenant. It might seem simple to deploy separate resources for every tenant, and it can work for a small numbers of tenants. However, it typically doesn't provide cost effectiveness, and it can become difficult to manage your resources. There are also various approaches that fit between these extremes, and they all have tradeoffs like scale, isolation, cost efficiency, performance, implementation complexity, and manageability.

Throughout this section, we discuss the main categories of Azure services that comprise a solution, including [compute](#), [storage and data](#), [networking](#), [deployment](#), [identity](#), [messaging](#), [artificial intelligence and machine learning](#), and [IoT](#). For each category, we outline the key patterns and approaches you can consider when you're designing a multitenant solution, and some antipatterns to avoid.

Deployment Stamps pattern

The [Deployment Stamps pattern](#) is frequently used in multitenant solutions. It involves deploying dedicated infrastructure for a tenant or for a group of tenants. A single stamp might contain multiple tenants or might be dedicated to a single tenant.



When using single-tenant stamps, the Deployment Stamps pattern tends to be straightforward to implement, because each stamp is likely to be unaware of any other, so no multitenancy logic or capabilities need to be built into the application layer. When

each tenant has their own dedicated stamp, this pattern provides the highest degree of isolation, and it mitigates the [Noisy Neighbor problem](#). It also provides the option for tenants to be configured or customized according to their own requirements, such as to be located in a specific geopolitical region or to have specific high availability requirements.

When using multitenant stamps, other patterns need to be considered to manage multitenancy within the stamp, and the Noisy Neighbor problem still might apply. However, by using the Deployment Stamps pattern, you can continue to scale as your solution grows.

The biggest problem with the Deployment Stamps pattern, when being used to serve a single tenant, tends to be the cost of the infrastructure. When using single-tenant stamps, each stamp needs to have its own separate set of infrastructure, which isn't shared with other tenants. You also need to ensure that the resources deployed for a stamp are sufficient to meet the peak load for that tenant's workload. Ensure that your [pricing model](#) offsets the cost of deployment for the tenant's infrastructure.

Single-tenant stamps often work well when you have a small number of tenants. As your number of tenants grows, it's possible but increasingly difficult to manage a fleet of stamps ([see this case study as an example](#)). You can also apply the Deployment Stamps pattern to create a fleet of multitenant stamps, which can provide benefits for resource and cost sharing.

To implement the Deployment Stamps pattern, it's important to use automated deployment approaches. Depending on your deployment strategy, you might consider managing your stamps within your deployment pipelines, by using declarative infrastructure as code, such as Bicep, ARM templates, or Terraform templates. Alternatively, you might consider building custom code to deploy and manage each stamp, such as by using [Azure SDKs](#).

Intended audience

The articles in this section are intended to be useful for solution architects and lead developers of multitenant applications, including independent software vendors (ISVs) and startups who develop SaaS solutions. Much of the guidance in this section is generic and applies to multiple Azure services within a category.

Next steps

We recommend you review the [approaches for resource organization in a multitenant solution](#) before reviewing the guidance about specific categories of Azure services.

Azure resource organization in multitenant solutions

Azure

Microsoft Entra ID

Azure provides many options for organizing your resources. In a multitenant solution, there are specific tradeoffs to consider, when you plan your resource organization strategy. In this article, we review two core elements of organizing your Azure resources: tenant isolation and scale-out across multiple resources. We also describe how to work with Azure's resource limits and quotas, and how to scale your solution beyond these limits.

Key considerations and requirements

Tenant isolation requirements

When you deploy a multitenant solution in Azure, you need to decide whether you dedicate resources to each tenant or share resources between multiple tenants. Throughout the multitenancy approaches and [service-specific guidance](#) sections of this series, we describe the options and trade-offs for many categories of resources. In general, there are a range of options for *tenant isolation*. Review [Tenancy models to consider for a multitenant solution](#) for more guidance about how to decide on your isolation model.

Scale

Most Azure resources, as well as resource groups and subscriptions, impose limits that can affect your ability to scale. You might need to consider *scaling out* or *bin packing* to meet your planned number of tenants or your planned system load.

If you know with certainty that you won't grow to large numbers of tenants or to a high load, don't overengineer your scale-out plan. But if you plan for your solution to grow, carefully consider your scale-out plan. Ensure that you architect for scale, by following the guidance in this article.

If you have an automated deployment process and need to scale across resources, determine how you'll deploy and assign tenants across multiple resource instances. For example, how will you detect that you're approaching the number of tenants that can be

assigned to a specific resource? Will you plan to deploy new resources *just in time* for when you need them? Or, will you deploy a pool of resources *ahead of time* so they're ready for you to use when you need them?

💡 Tip

In the early stages of design and development, you might not choose to implement an automated scale-out process. You should still consider and clearly document the processes required to scale as you grow.

It's also important to avoid making assumptions in your code and configuration, which can limit your ability to scale. For example, you might need to scale out to multiple storage accounts. Ensure your application tier doesn't assume that it only connects to a single storage account for all tenants.

Approaches and patterns to consider

Tenant isolation

Azure resources are deployed and managed through a hierarchy. Most *resources* are deployed into *resource groups*, which are contained in *subscriptions*. *Management groups* logically group subscriptions together. All of these hierarchical layers are associated with a *Microsoft Entra tenant*.

When you determine how to deploy resources for each tenant, you might isolate at different levels in the hierarchy. Each option is valid for certain types of multitenant solutions, and comes with benefits and tradeoffs. It's also common to combine approaches, using different isolation models for different components of a solution.

Isolation within a shared resource

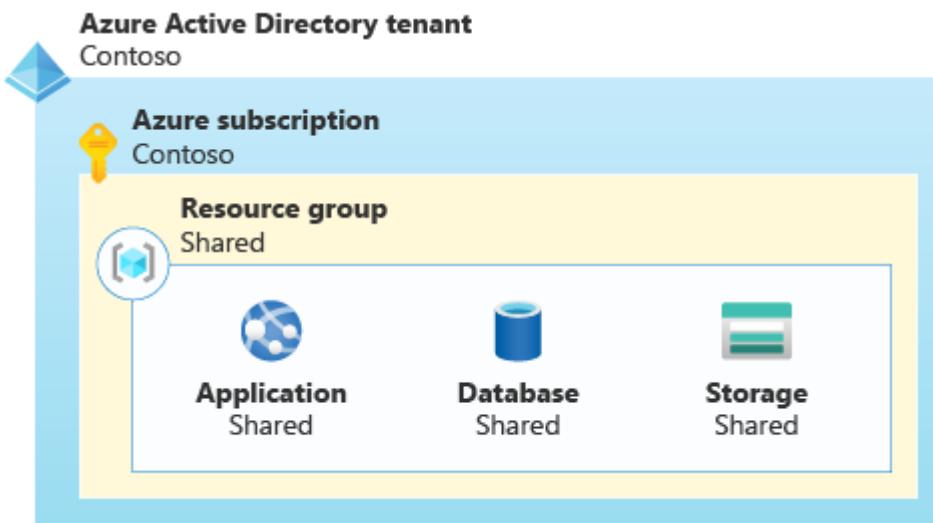
You might choose to share an Azure resource among multiple tenants, and run all of their workloads on a single instance. Review the [service-specific guidance](#) for the Azure services you use to understand any specific considerations or options that might be important.

When you run single instances of a resource, you need to consider any service limits, subscription limits, or quotas that might be reached as you scale. For example, there's a maximum number of nodes that are supported by an Azure Kubernetes Service (AKS) cluster, and there's an upper limit on the number of transactions per second that are

supported by a storage account. Consider how you'll [scale to multiple shared resources](#) as you approach these limits.

You also need to ensure your application code is fully aware of multitenancy, and that it restricts access to the data for a specific tenant.

As an illustration of the shared resource approach, suppose Contoso is building a multitenant SaaS application that includes a web application, a database, and a storage account. They might decide to deploy shared resources, and they'd use these resources to service all of their customers. In the following diagram, a single set of resources is shared by all the customers.



Separate resources in a resource group

You can also deploy dedicated resources for each tenant. You might deploy an entire copy of your solution for a single tenant. Or, you might share some components between tenants and deploy other components that are dedicated to a specific tenant.

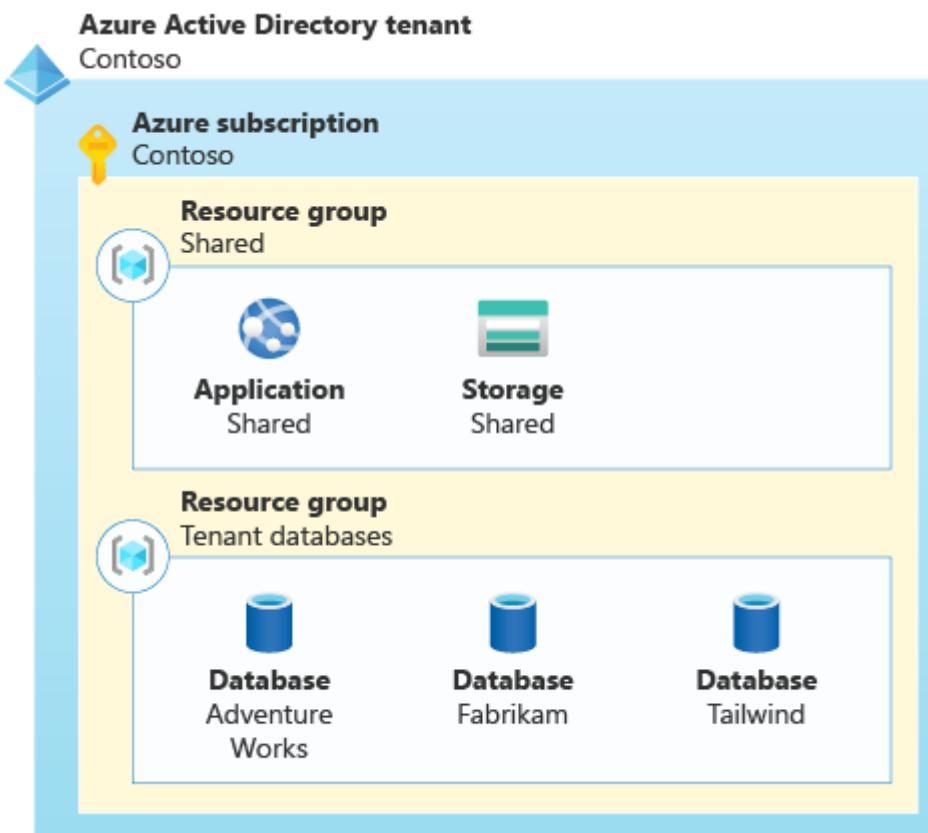
We recommend that you use resource groups to manage resources with the same lifecycle. In some multitenant systems, it makes sense to deploy resources for multiple tenants into a single resource group or a set of resource groups.

It's important that you consider how you deploy and manage these resources, including [whether the deployment of tenant-specific resources is initiated by your deployment pipeline or your application](#). You also need to determine how you'll [clearly identify that specific resources relate to specific tenants](#). Consider using a clear [naming convention strategy, resource tags](#), or a tenant catalog database.

It's a good practice to use separate resource groups for the resources you share between multiple tenants and the resources that you deploy for individual tenants. However, for some resources, [Azure limits the number of resources of a single type that](#)

can be deployed into a resource group. This limit means you might need to [scale across multiple resource groups](#) as you grow.

Suppose Contoso has three customers: Adventure Works, Fabrikam, and Tailwind. They might choose to share the web application and storage account between the three customers, and then deploy individual databases for each tenant. In the following diagram, each customer is assigned a resource group that contains shared resources and a resource group that contains a database.



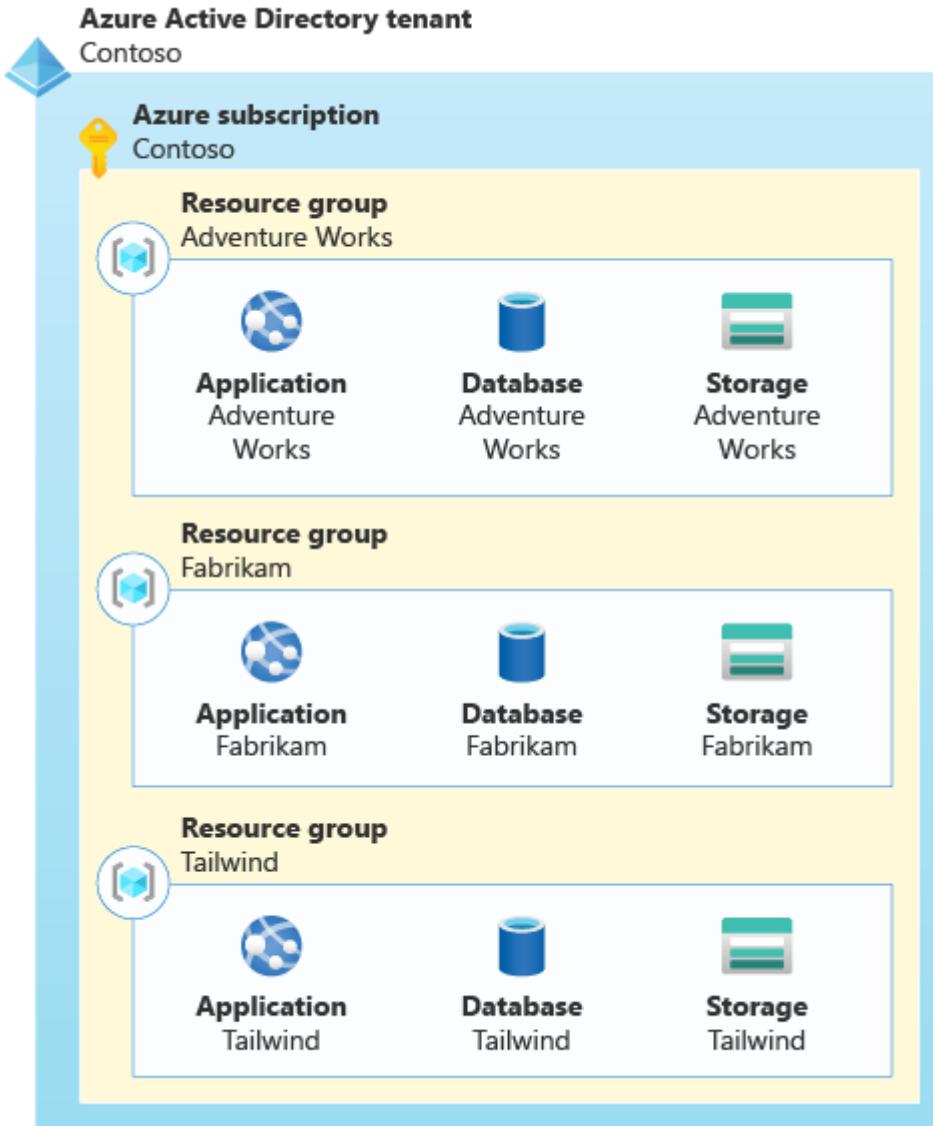
Separate resource groups in a subscription

When you deploy a set of resources for each tenant, consider using dedicated tenant-specific resource groups. For example, when you follow the [Deployment Stamps pattern](#), each stamp should be deployed into its own resource group. You can consider deploying multiple tenant-specific resource groups into a shared Azure subscription, which enables you to easily configure policies and access control rules.

You might choose to create a set of resource groups for each tenant, and also shared resource groups for any shared resources.

When you deploy tenant-specific resource groups into shared subscriptions, be aware of the maximum number of resource groups in each subscription, and other subscription-level limits that apply to the resources you deploy. As you approach these limits, you might need to [scale across multiple subscriptions](#).

In our example, Contoso might choose to deploy a stamp for each of their customers and place the stamps in dedicated resource groups within a single subscription. In the following diagram, a subscription, which contains three resource groups, is created for each customer.



Separate subscriptions

By deploying tenant-specific subscriptions, you can completely isolate tenant-specific resources. Additionally, because most quotas and limits apply within a subscription, using a separate subscription per tenant ensures that each tenant has full use of any applicable quotas. For some Azure billing account types, [you can programmatically create subscriptions](#). You can also use [Azure reservations](#) and [Azure savings plan for compute](#) across subscriptions.

Make you are aware of the number of subscriptions that you can create. The maximum number of subscriptions might differ, depending on your commercial relationship with Microsoft or a Microsoft partner, such as if you have an [enterprise agreement](#).

However, it can be more difficult to request quota increases, when you work across a large number of subscriptions. The [Quota API](#) provides a programmatic interface for some resource types. However, for many resource types, quota increases must be requested by [initiating a support case](#). It can also be challenging to work with Azure support agreements and support cases, when you work with many subscriptions.

Consider grouping your tenant-specific subscriptions into a [management group](#) hierarchy, to enable easy management of access control rules and policies.

For example, suppose Contoso decided to create separate Azure subscriptions for each of their three customers, as shown in the following diagram. Each subscription contains a resource group, with the complete set of resources for that customer.



Each subscription contains a resource group, with the complete set of resources for that customer.

They use a management group to simplify the management of their subscriptions. By including *Production* in the management group's name, they can clearly distinguish any production tenants from non-production or test tenants. Non-production tenants would have different Azure access control rules and policies applied.

All of their subscriptions are associated with a single Microsoft Entra tenant. Using a single Microsoft Entra tenant means that the Contoso team's identities, including users and service principals, can be used throughout their entire Azure estate.

Separate subscriptions in separate Microsoft Entra tenants

It's also possible to manually create individual Microsoft Entra tenants for each of your tenants, or to deploy your resources into subscriptions within your customers' Microsoft Entra tenants. However, working with multiple Microsoft Entra tenants makes it more difficult to authenticate, to manage role assignments, to apply global policies, and to perform many other management operations.

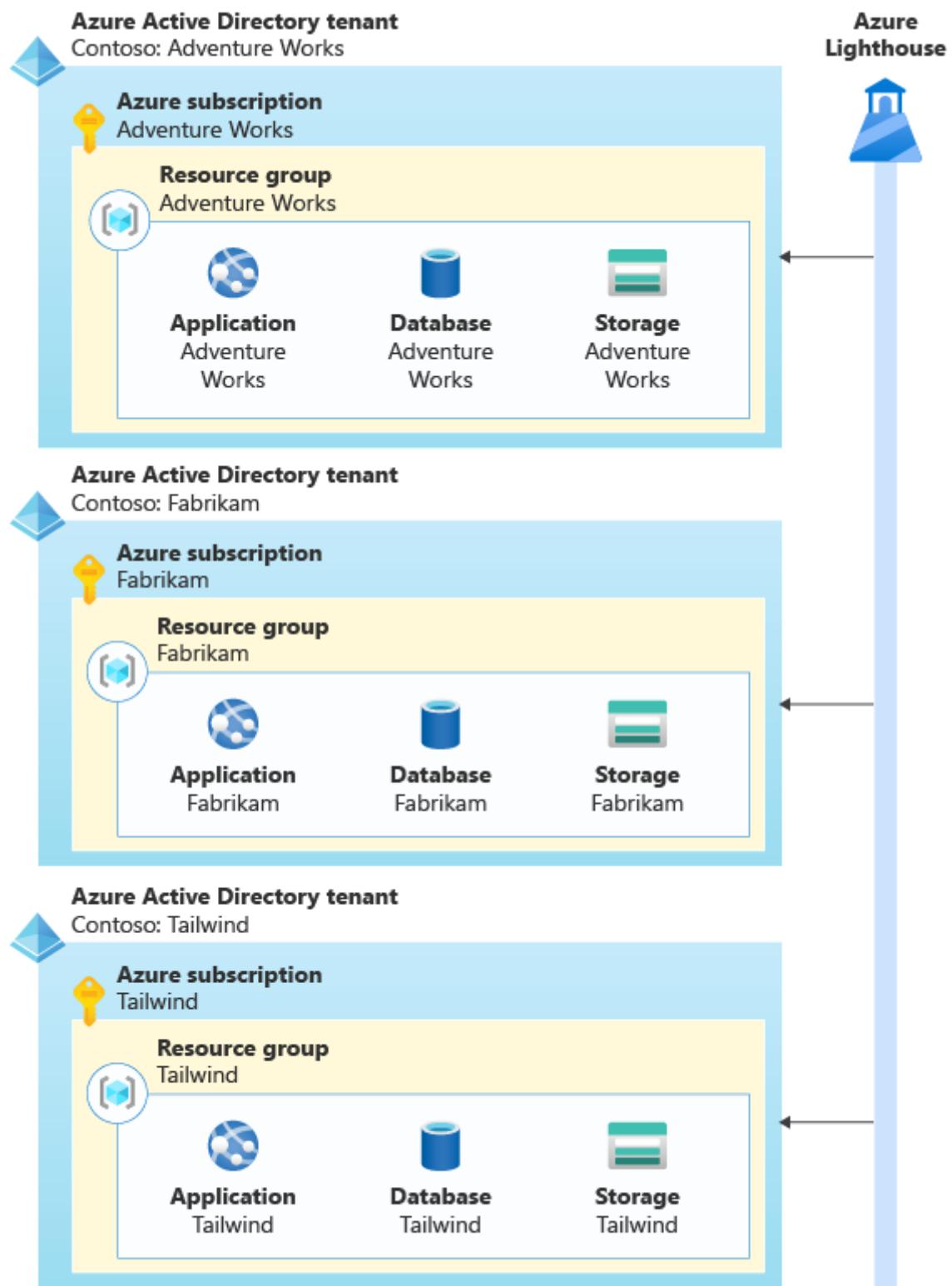
Warning

We advise against creating multiple Microsoft Entra tenants for most multitenant solutions. Working across Microsoft Entra tenants introduces extra complexity and reduces your ability to scale and manage your resources. Typically, this approach is only used by managed service providers (MSPs), who operate Azure environments on behalf of their customers.

A single Microsoft Entra tenant can be used by multiple separate subscriptions and Azure resources. Before you make an effort to deploy multiple Microsoft Entra tenants, consider whether there are other approaches that could achieve your purposes [↗].

In situations where you need to manage Azure resources in subscriptions that are tied to multiple Microsoft Entra tenants, consider using [Azure Lighthouse](#) to help manage your resources across your Microsoft Entra tenants.

For example, Contoso could create separate Microsoft Entra tenants and separate Azure subscriptions for each of their customers, as shown in the following diagram.



A Microsoft Entra tenant is configured for each of Contoso's tenants, which contains a subscription and the resources required. Azure Lighthouse is connected to each Microsoft Entra tenant.

Bin packing

Regardless of your resource isolation model, it's important to consider when and how your solution will scale out across multiple resources. You might need to scale your

resources, as the load on your system increases, or as the number of tenants grows. Consider *bin packing* to deploy an optimal number of resources for your requirements.

💡 Tip

In many solutions, it's easier to scale your entire set of resources together, instead of scaling resources individually. Consider following the **Deployment Stamps pattern**.

Resource limits

Azure resources have [limits and quotas](#) that must be considered in your solution planning. For example, resources might support a maximum number of concurrent requests or tenant-specific configuration settings.

The way you configure and use each resource also affects the scalability of that resource. For example, given a certain amount of compute resources, your application can successfully respond to a defined number of transactions per second. Beyond this point, you might need to scale out. Performance testing helps you to identify the point at which your resources no longer meet your requirements.

ⓘ Note

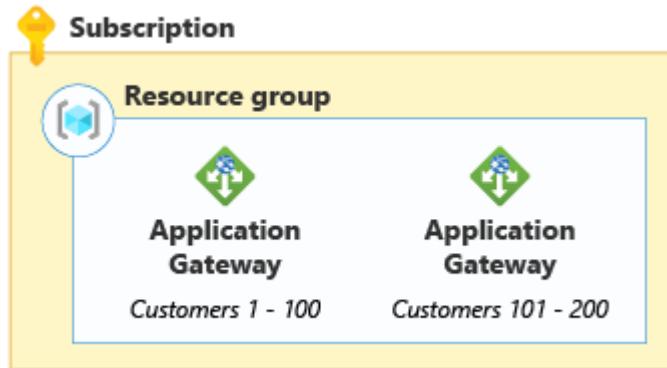
The principle of scaling to multiple resources applies even when you work with services that support multiple instances.

For example, Azure App Service supports scaling out the number of instances of your plan, but there are limits for how far you can scale a single plan. In a high-scale multitenant app, you might exceed these limits and need to deploy additional App Service resources to match your growth.

When you share some of your resources between tenants, you should first determine the number of tenants that the resource supports, when it's configured according to your requirements. Then, deploy as many resources as you need to serve your total number of tenants.

For example, suppose you deploy an Azure Application Gateway, as part of a multitenant SaaS solution. You review your application design, test the application gateway's performance under load, and review its configuration. Then, you determine that a single application gateway resource can be shared among 100 customers. According to your organization's growth plan, you expect to onboard 150 customers in

your first year, so you need to plan to deploy multiple application gateways to service your expected load.

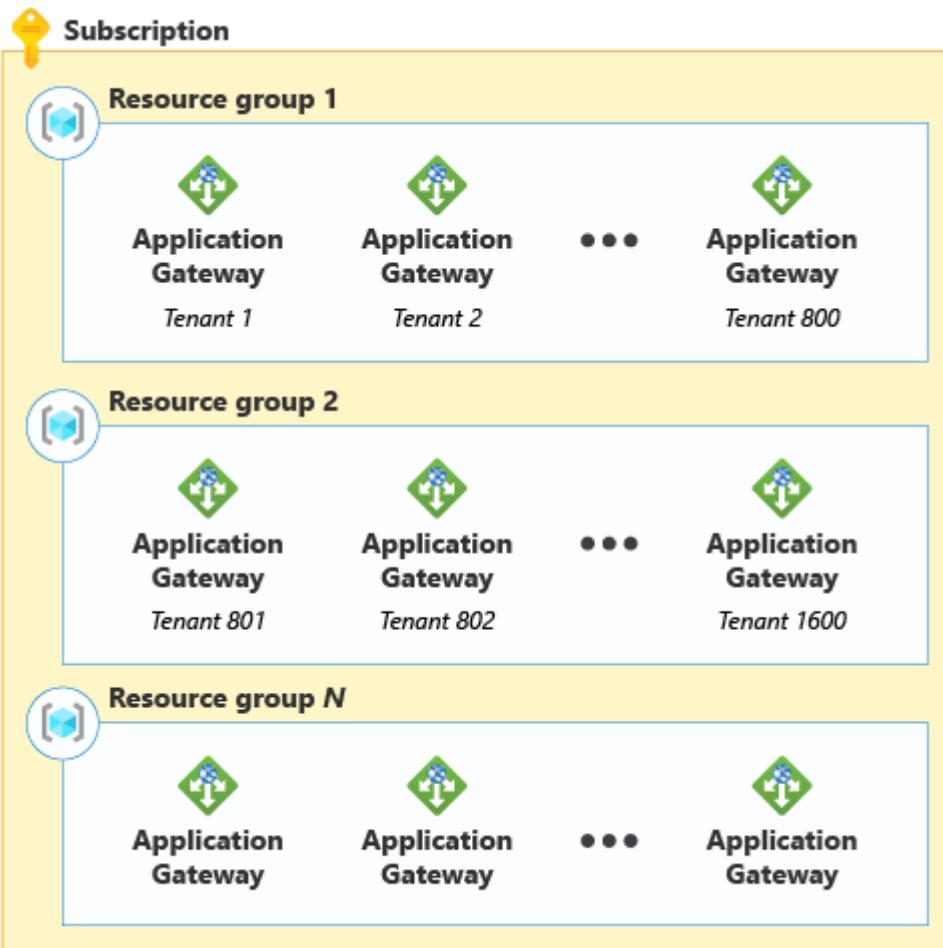


In the previous diagram, there are two application gateways. The first gateway is dedicated to customers 1 through 100, and the second is dedicated to customers 101 through 200.

Resource group and subscription limits

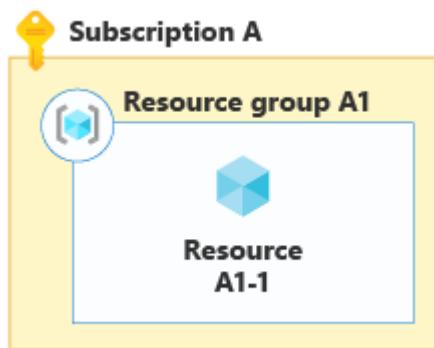
Whether you work with shared or dedicated resources, it's important to account for limits. Azure limits the number of resources that can be [deployed into a resource group](#) and [into an Azure subscription](#). As you approach these limits, you need to plan to scale across multiple resource groups or subscriptions.

For example, suppose you deploy a dedicated application gateway, for each of your customers, into a shared resource group. For some resources, [Azure supports deploying up to 800 resources of the same type](#) into a single resource group. So, when you reach this limit, you need to deploy any new application gateways into another resource group. In the following diagram, there are two resource groups. Each resource group contains 800 application gateways.

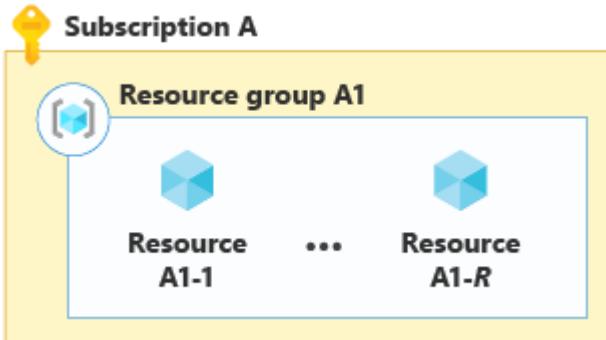


Bin pack tenants across resource groups and subscriptions

You can also apply the bin packing concept across resources, resource groups, and subscriptions. For example, when you have a small number of tenants you might be able to deploy a single resource and share it among all of your tenants. The following diagram shows bin packing into a single resource.



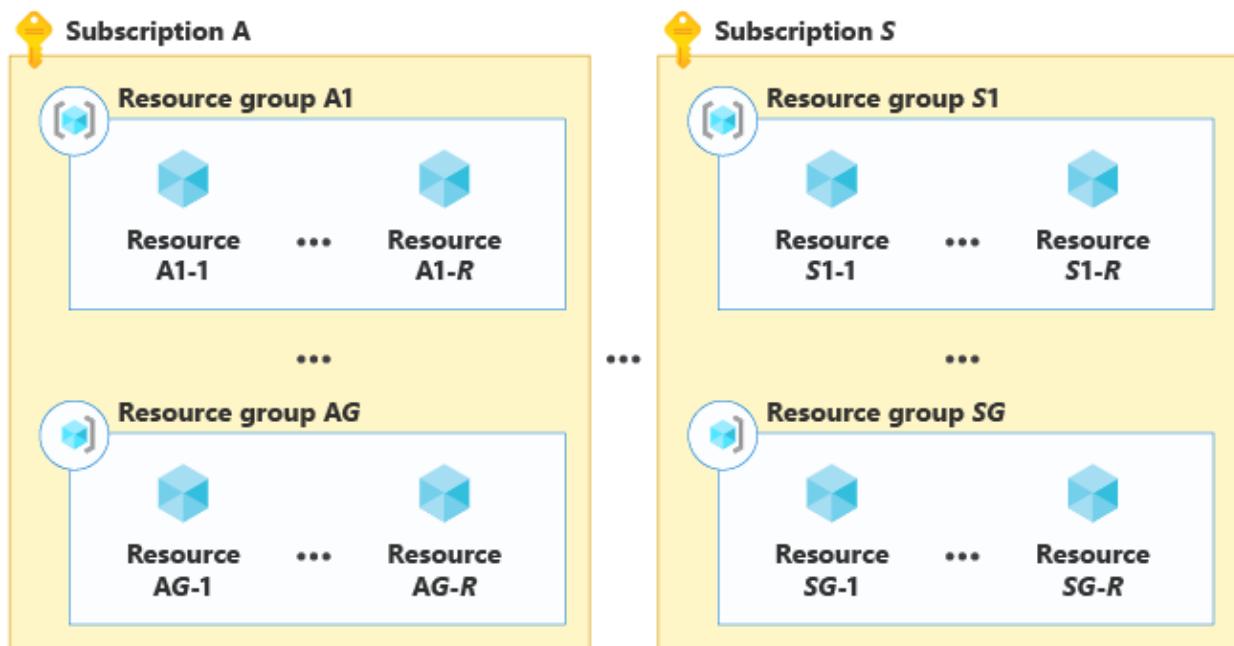
As you grow, you might approach the capacity limit for a single resource, and scale out to multiple (R) resources. The following diagram shows bin packing across multiple resources.



Over time, you might reach the limit of the number of resources in a single resource group, and you would then deploy multiple (R) resources into multiple (G) resource groups. The following diagram shows bin packing across multiple resources, in multiple resource groups.



And as you grow even larger, you can deploy across multiple (S) subscriptions, each containing multiple (G) resource groups with multiple (R) resources. The following diagram shows bin packing across multiple resources, in multiple resource groups and subscriptions.



By planning your scale-out strategy, you can scale to extremely large numbers of tenants and sustain a high level of load.

Tags

Resource tags enable you to add custom metadata to your Azure resources, which can be useful for management and tracking costs. For more details, see [Allocate costs by using resource tags](#).

Antipatterns to avoid

- **Not planning for scale.** Ensure you have a clear understanding of the limits of the resources you'll deploy, and which limits might become important, as your load or number of tenants increase. Plan how you'll deploy additional resources as you scale, and test the plan.
- **Not planning to bin pack.** Even if you don't need to grow immediately, plan to scale your Azure resources across multiple resources, resource groups, and subscriptions over time. Avoid making assumptions in your application code, like there being a single resource when you might need to scale to multiple resources in the future.
- **Scaling many individual resources.** If you have a complex resource topology, it can become difficult to scale individual components, one by one. It's often simpler to scale your solution as a unit, by following the [Deployment Stamps pattern](#).
- **Deploying isolated resources for each tenant, when not required.** In many solutions, it's more cost effective and efficient to deploy shared resources for multiple tenants.

- **Using separate Microsoft Entra tenants.** In general, it's inadvisable to provision multiple Microsoft Entra tenants. Managing resources across Microsoft Entra tenants is complex. It's simpler to scale across subscriptions linked to a single Microsoft Entra tenant.
- **Overarchitecting when you don't need to scale.** In some solutions, you know with certainty that you'll never grow beyond a certain level of scale. In these scenarios, there's no need to build complex scaling logic. However, if your organization plans to grow, then you will need to be prepared to scale—potentially at short notice.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Jason Beck](#) | Senior Customer Engineer, FastTrack for Azure
- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure
- [Laura Nicolas](#) | Senior Customer Engineer, FastTrack for Azure
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [Joshua Waddell](#) | Senior Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [Cost management and allocation](#) approaches.

Architectural approaches for governance and compliance in multitenant solutions

Article • 03/20/2023

As your use of Azure matures, it's important to consider the governance of your cloud resources. Governance includes how tenants' data is stored and managed, and how you organize your Azure resources. You might also need to follow regulatory, legal, or contractually mandated standards. This article provides information about how to consider governance and compliance in a multitenant solution. It also suggests some of the key Azure platform features that support these concerns.

Key considerations and requirements

Resource isolation

Ensure you configure your Azure resources to meet your tenants' isolation requirements. See [Azure resource organization in multitenant solutions](#) for guidance on isolating your Azure resources.

Data management

When you store data on behalf of your tenants, you might have requirements or obligations that you need to meet. From a tenant's perspective, they often expect ownership and control of their data. Consider how you isolate, store, access, and aggregate tenants' data. Uncover tenants' expectations and requirements that could affect how your solution works.

Isolation

Review the [Architectural approaches for storage and data in multitenant solutions](#) to understand how to isolate tenants' data. Consider whether tenants have requirements to use their own data encryption keys.

Whichever isolation approaches you implement, be prepared for tenants to request an audit of their data. It's a good practice to document all of the data stores in which tenants' data might be kept. Common data sources include the following:

- Databases and storage accounts deployed as part of your solution.
- Identity systems, which are often shared between tenants.
- Logs.
- Data warehouses.

Sovereignty

Understand whether there are any restrictions on the physical location for your tenants' data that's to be stored or processed. Your tenants might require you store their data in specific geographic locations. They might also require that you *don't* store their data in certain locations. Although these requirements are commonly based on legislation, they can also be based on cultural values and norms.

For more information about data residency and sovereignty, see the whitepaper [Enabling Data Residency and Data Protection in Microsoft Azure Regions](#).

Tenants' access to data that you store

Tenants sometimes request direct access to the data you store on their behalf. For example, they might want to ingest their data into their own data lake.

Plan how you'll respond to these requests. Consider whether any of the tenants' data is kept in shared data stores. If it is, plan how you'll avoid tenants accessing other tenants' data.

Avoid providing direct access to databases or storage accounts unless you designed for this requirement, such as by using the [Valet Key pattern](#). Consider creating an API or automated data export process for integration purposes.

For more information about integration with tenants' systems, and external systems, see [Architectural approaches for tenant integration and data access](#).

Your access to tenants' data

Consider whether your tenants' requirements restrict the personnel who can work with their data or resources. For example, suppose you build a SaaS solution that's used by many different customers. A government agency might require that only citizens of their country/region are allowed to access the infrastructure and data for their solution. You might meet this requirement by using separate Azure resource groups, subscriptions, or management groups for sensitive customer workloads. You can apply tightly scoped Azure role-based access controls (RBAC) role assignments for specific groups of users to work with these resources.

Aggregation of data from multiple tenants

Consider whether you need to combine or aggregate data from multiple tenants. For example, do you analyze the aggregated data, or train machine learning models that could be applied to other tenants? Ensure your tenants understand the ways in which you use their data. Include any use of aggregated or anonymized data.

Compliance requirements

It's important that you understand whether you need to meet any compliance standards. Compliance requirements might be introduced in several situations, including:

- You, or any of your tenants, work within certain industries. For example, if any of your tenants work in the [Healthcare industry](#), you might need to comply with the HIPAA standard.
- You, or any of your tenants, are located in geographic or geopolitical regions that require compliance with local laws. For example, if any of your tenants are located in Europe, you might need to comply with [General Data Protection Regulation \(GDPR\)](#).
- You purchase a cyberinsurance policy to mitigate the risk of breaches. Cyberinsurance providers might require that you follow their standards and apply specific controls for your policy to be valid.

Important

Compliance is a shared responsibility between Microsoft, you, and your tenants.

Microsoft ensures that our services meet a specific set of compliance standards, and provides tools like [Microsoft Defender for Cloud](#) that help to verify your resources are configured according to those standards.

However, ultimately it is your responsibility to fully understand the compliance requirements that apply to your solution, and how to configure your Azure resources according to those standards. See [Azure compliance offerings](#) for more detail.

This article doesn't provide specific guidance about how to become compliant with any particular standards. Instead, it provides some general guidance around how to consider compliance and governance in a multitenant solution.

If different tenants need you to follow different compliance standards, plan to comply with the most stringent standard across your entire environment. It's easier to follow one strict standard than to follow different standards for different tenants.

Approaches and patterns to consider

Resource tags

Use [resource tags](#) to track the tenant identifier for tenant-specific resources, or the stamp identifier when you scale using the [Deployment Stamps pattern](#). By using resource tags, you can quickly identify resources that are associated with specific tenants or stamps.

Access control

Use [Azure RBAC](#) to restrict access to the Azure resources that constitute the multitenant solution. Follow the RBAC [best practices](#), such as applying role assignments to groups instead of users. Scope your role assignments so they provide the minimum permissions necessary. Avoid long-standing access to resources by using just-in-time access and features like [Azure Active Directory Privileged Access Management](#).

Azure Resource Graph

[Azure Resource Graph](#) enables you to work with Azure resource metadata. By using Resource Graph, you can query across a large number of Azure resources, even if they're spread across multiple subscriptions. Resource Graph can query for the resources of a specific type, or to identify resources that have been configured in specific ways. It can also be used to track the history of a resource's configuration.

Resource Graph can be helpful to manage large Azure estates. For example, suppose you deploy tenant-specific Azure resources across multiple Azure subscriptions. By [applying tags to your resources](#), you can use the Resource Graph API to find resources that are used by specific tenants or deployment stamps.

Azure Purview

Consider using [Azure Purview](#) to track and classify the data that you store. When tenants request access to their data, you can easily determine the data sources that you should include.

Verify compliance with standards

Use tools like [Azure Policy](#), [Microsoft Defender for Cloud's regulatory compliance portal](#), and [Azure Advisor](#). These tools help you to configure your Azure resources to meet compliance requirements and to follow the recommended best practices.

Generate compliance documentation

Your tenants might require that you demonstrate your compliance with specific standards. Use the [Service Trust Portal](#) to generate compliance documentation that you can provide to your tenants or to third-party auditors.

Some multitenant solutions incorporate Microsoft 365 and use services like Microsoft OneDrive, Microsoft SharePoint, and Microsoft Exchange Online. The [Microsoft Purview compliance portal](#) helps you understand how these services comply with regulatory standards.

Deployment Stamps pattern

Consider following the [Deployment Stamps pattern](#) when you need to comply with tenant-specific requirements.

For example, you might deploy stamps of your solution into multiple Azure regions. Then, you can assign new tenants to stamps, based on the regions that they need to have their data located in.

Similarly, a new tenant might introduce strict compliance requirements that you can't meet within your existing solution components. You can consider deploying a dedicated stamp for that tenant, and then configure it according to their requirements.

Antipatterns to avoid

- **Not understanding your tenants' compliance requirements.** It's important not to make assumptions about the compliance requirements that your tenants might impose. If you plan to grow your solution into new markets, be mindful of the regulatory environment that your tenants are likely to operate within.
- **Ignoring good practices.** If you don't have any immediate need to adhere to compliance standards, you should still follow good practices when you deploy your Azure resources. For example, isolate your resources, apply policies to verify resource configuration, and apply role assignments to groups instead of users. By

following good practices, you make it simpler to follow compliance standards when you eventually need to do so.

- **Assuming there are no compliance requirements.** When you first launch a multitenant solution, you might not be aware of compliance requirements, or you might not need to follow any. As you grow, you'll likely need to provide evidence that you comply with various standards. Use [Microsoft Defender for Cloud](#) to monitor your compliance posture, even before you have an explicit requirement to do so.
- **Not planning for management.** As you deploy your Azure resources, consider how you plan to manage them. If you need to make bulk updates to resources, ensure you have an understanding of automation tools, such as the Azure CLI, Azure PowerShell, Azure Resource Graph, and the Azure Resource Manager APIs.
- **Not using management groups.** Plan your subscription and management group hierarchy, including access control and Azure Policy resources at each scope. It can be difficult and disruptive to introduce or change these elements when your resources are used in a production environment.
- **Failing to plan your access control strategy.** Azure RBAC provides a high degree of control and flexibility in how you manage access to your resources. Ensure you use Azure AD groups to avoid assigning permissions to individual users. Assign roles at scopes that provide an appropriate balance between security and flexibility. Use built-in role definitions wherever possible, and assign roles that provide the minimum permissions required.
- **Not using Azure Policy.** It's important to use Azure Policy to govern your Azure environment. After you plan and deploy policies, ensure you monitor the policy compliance and carefully review any violations or exceptions.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure
- [Laura Nicolas](#) | Senior Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [approaches for cost management and allocation](#).

Architectural approaches for cost management and allocation in a multitenant solution

Azure

Azure Cost Management

Azure Resource Manager

Azure Monitor

Multitenant solutions often require special consideration when you measure and allocate costs, and when you optimize costs. On this page, we describe some key guidance for solution architects to consider about the measurement, allocation, and optimization of costs for a multitenant application.

Key considerations and requirements

Consider the requirements you have for measuring the consumption for your solution. This is discussed in more detail on [Measure the consumption of each tenant](#).

Purpose of measurement

It's important to decide what your goal is. The following are examples of goals:

- **Calculate an approximate cost of goods sold for each tenant.** For example, if you deploy a significant number of shared resources, you might only be interested in a rough approximation of the cost incurred for each tenant.
- **Calculate the exact cost incurred by each tenant.** For example, if you charge your tenants for the exact amount of consumption they incur, you need to have precise information about how much each tenant's resources cost.
- **Identify outlier tenants that cost significantly more than others.** For example, if you provide a [flat-rate pricing model](#), you might need to determine whether any tenants are consuming a disproportionate amount of your provisioned capacity, so that you can apply fair-use policies. In many situations, this use case doesn't require precise measurement of costs.
- **Reduce the overall Azure cost for your solution.** For example, you might want to look at the cost of every component, and then determine whether you have over-provisioned for the workload.

By understanding the goal of measuring the consumption by a tenant, you can determine whether the cost allocations need to be approximate or highly precise, which affects the specific tools you can use and the practices you can follow.

Shared components

You might be able to reduce the cost of a multitenant solution by moving tenants to shared infrastructure. However, you need to carefully consider the impact of sharing resources, such as whether your tenants will begin to experience the [Noisy Neighbor problem](#).

You also need to consider how you measure and allocate the costs of shared components. For example, you can evenly divide the cost between each of the tenants that use the shared component. Or, you can meter each tenant's usage to get a more precise measurement of their consumption of shared components.

Approaches and patterns to consider

Allocate costs by using resource tags

Azure enables you to [apply tags to your resources](#). A tag is a key-value pair. You use tags to add custom metadata. Tags are useful for many management operations, and they're also useful for analyzing the cost of your Azure consumption. After you apply tags, [you can determine costs associated with each tag](#).

The way you use tags in a multitenant solution is likely to be different, depending on your architecture.

In some solutions, you might deploy dedicated resources for each tenant, such as if you deploy dedicated [Deployment Stamps](#) for each tenant. In these situations, it's clear that any Azure consumption for those resources should be allocated to that tenant, and so you can tag your Azure resources with the tenant ID.

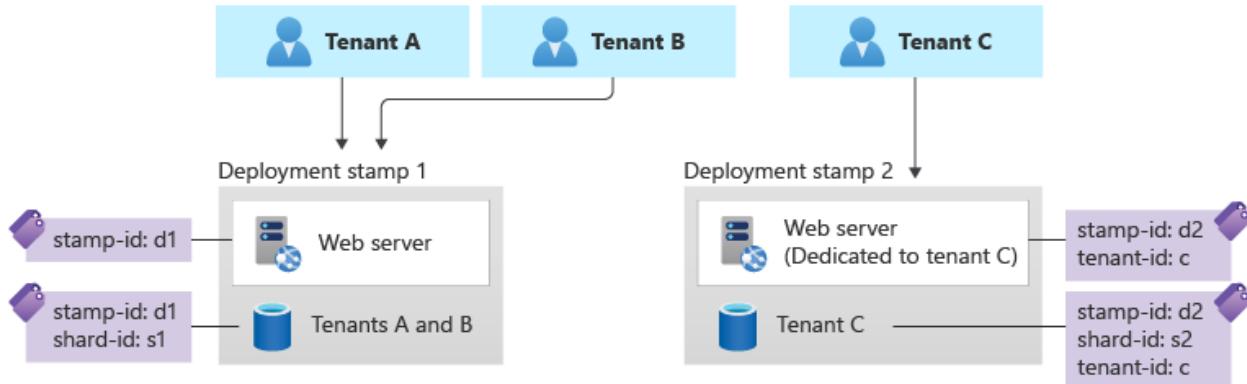
In other situations, you might have sets of shared resources. For example, when you apply the [Sharding pattern](#), you might deploy multiple databases and spread your tenants across them. Consider tagging the resources with an identifier for the *group* of tenants. You might not be able to easily allocate costs to a single tenant, but you can at least narrow down the cost to a set of tenants, when you use this approach. You can also use the consumption information to help you rebalance tenants across the shards, if you notice that a specific shard is accruing higher costs than the others.

ⓘ Note

There is a [limit to the number of tags](#) that can be applied to a resource. When you work with shared resources, it's best not to add a tag for every tenant that shares

the resource. Instead, consider adding a tag with the shard ID or another way to identify the group of tenants.

Consider an example multitenant solution that's built using the [Deployment Stamps pattern](#) and a [vertically partitioned tenancy model](#). Each deployment stamp includes a shared web server and sharded databases. Tags can be applied to each of the Azure components, as shown in the following diagram.



The tagging strategy employed here is as follows:

- Every resource has a `stamp-id` tag.
- Every sharded database has a `shard-id` tag.
- Every resource dedicated to a specific tenant has a `tenant-id` tag.

With this tagging strategy, it's easy to filter the cost information to a single stamp. It's also easy to find the cost of the tenant-specific resources, such as the total cost of the database for tenant C. Shared components don't have a `tenant-id` tag, but the cost of the shared components for a stamp can be divided between the tenants who are assigned to use that stamp or shard.

Instrument your application

In situations where you don't have a direct relationship between an Azure resource and a tenant, consider instrumenting your application to collect telemetry.

Your application tier might already collect logs and metrics that are helpful to answer questions about metering, for example:

- Approximately how many API requests are made per tenant?
- What times of the day are specific tenants busiest?
- How do tenant A's usage patterns compare to tenant B's usage patterns?

In Azure, these metrics are often captured by [Application Insights](#). By using [telemetry initializers](#), you can enrich the telemetry captured by Application Insights, to include a tenant identifier or other custom data.

However, Application Insights and other logging and monitoring solutions are not appropriate for precise cost measurement or for metering purposes. Application Insights is designed to [sample data](#), especially when your application has a high volume of requests. Sampling is designed to reduce the cost of monitoring your solution, because capturing every piece of telemetry can often become expensive.

If you need to track precise details about consumption or usage for billing purposes, you should instead build a custom pipeline to log the necessary data. You should then aggregate the data, based on your requirements. Azure services that can be helpful for this purpose include [Event Hubs](#), to capture large volumes of telemetry, and [Stream Analytics](#), to process it in real time.

Use Azure Reservations and Azure savings plan to reduce costs

Azure Reservations: [Azure Reservations](#) enable you to reduce your Azure costs by pre-committing to a certain level of spend. Reservations apply to a number of Azure resource types.

Reservations can be used effectively in a multitenant solution. Note the following considerations:

- When you deploy a multitenant solution that includes shared resources, consider the baseline level of consumption that you need for the workload. You might consider a reservation for that baseline consumption, and then you'd pay standard rates for higher consumption during unpredictable peaks.
- When you deploy resources for each tenant, consider whether you can pre-commit to the resource consumption for a specific tenant, or across your portfolio of tenants.

Azure Reservations enables you to [scope your reservations](#) to apply to a resource group, a subscription, or a set of subscriptions. This means that you can take advantage of reservations, even if you shard your workload across multiple subscriptions.

Reservation scopes can also be helpful, when you have tenants with unpredictable workloads. For example, consider a solution in which tenant A only needs one instance of a specific resource, but tenants B and C each need two. Then tenant B becomes less

busy, so you reduce the instance count, and tenant A gets busier, so you increase the instance count. Your reservations are applied to the tenants that need them.

Azure savings plan for compute: Azure savings plan for compute is a flexible cost-saving plan that generates significant savings over pay-as-you-go prices. You agree to a one-year or three-year contract and receive discounts on eligible compute services. These services include virtual machines, dedicated hosts, container instances, Azure premium functions, and Azure app services. Savings apply to these compute services regardless of the region, instance size, or operating system. For more information, see [Azure savings plan overview](#) and [Azure savings plan documentation](#).

Combine reservations and savings plan: To further optimize cost and flexibility, you can combine an Azure savings plan with Azure Reservations.

Antipatterns to avoid

- **Not tracking costs at all.** It's important to have at least an approximate idea of the costs you're incurring, and how each tenant impacts the cost of delivering your solution. Otherwise, if your costs change over time, you have no baseline to compare against. You also might not be able to predict how a growth in tenants will impact your costs and profitability.
- **Making assumptions or guessing.** Ensure your cost measurement is based on real information. You might not need a high degree of precision, but even your estimates should be informed by real measurements.
- **Unnecessary precision.** You might not need to have a detailed accounting of every cost that's incurred for every tenant. Building unnecessarily precise cost measurement and optimization processes can be counterproductive, because it adds engineering complexity and creates brittle processes.
- **Real-time measurement.** Most solutions don't need up-to-the-minute cost measurements. Because metering and consumption data can be complex to process, you should log the necessary data and then asynchronously aggregate and process the data later.
- **Using monitoring tools for billing.** As described in [Instrument your application](#), ensure you use tools that are designed for cost monitoring and metering. Application monitoring solutions are typically not good candidates for this type of data, especially when you need high precision.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Sherri Babylon](#) | Senior Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Measure the consumption of each tenant

Architectural approaches for the deployment and configuration of multitenant solutions

Azure Azure DevOps Azure Pipelines Azure Marketplace GitHub

Regardless of your architecture and the components you use to implement it, you need to deploy and configure your solution's components. In a multitenant environment, it's important to consider how you deploy your Azure resources, especially when you deploy dedicated resources for each tenant, or when you reconfigure resources based on the number of tenants in your system. On this page, we provide solution architects with guidance about deploying multitenant solutions, and we demonstrate some approaches to consider when you plan your deployment strategy.

Key considerations and requirements

It's important to have a clear idea of your requirements, before you plan your deployment strategy. Specific considerations include the following:

- **Expected scale:** Do you expect to support a small number of tenants (such as five or less), or will you grow to a large number of tenants?
- **Automated or supported onboarding:** When a tenant is ready to be onboarded, will they be able to complete the process themselves by following an automated procedure? Or, do new tenants initiate a request, and you manually onboard them after you receive the request? Are there manual approval steps required from your team, such as to prevent the abuse of your service?
- **Provisioning time:** When a tenant is ready to be onboarded, how quickly does the onboarding process need to be completed? If you don't have a clear answer, consider whether this should be measured in seconds, minutes, hours, or days.
- **Azure Marketplace:** Do you plan to use the Azure Marketplace to initiate the deployment? If you do, there are [requirements that you need to meet to add new tenants](#).

You should also consider onboarding and provisioning steps, automation, and resource management responsibility.

Onboarding and provisioning steps

Consider everything that you need to do when onboarding a tenant, and document this list and workflow, even if it's performed manually. The onboarding workflow typically includes the following:

- Acceptance of commercial agreements.
- Collection of the information that you need to configure your system for the new tenant.
- Manual approval steps, for example, to prevent fraud or abuse of your service.
- The provisioning of resources in Azure.
- [Creating or configuring domain names](#).
- Perform post-deployment configuration tasks, such as creating the first user account for the tenant and securely transmitting its credentials to the tenant.
- Manual configuration changes, such as DNS record changes.

Clearly document the workflow that's required to onboard a new tenant.

Additionally, consider the specific Azure resources that you need to provision for a tenant. Even if you don't provision dedicated resources for each tenant, consider whether you sometimes need to deploy resources when a new tenant is onboarded. This might occur when a tenant requires their data to be stored in a specific region, or when you approach the limits of a stamp or component in your solution and need to create another instance for the next batch of tenants.

Consider whether the onboarding process is likely to be disruptive to other tenants, especially to those who share the same infrastructure. For example, if you need to modify shared databases, could this process cause a performance impact that other tenants might notice? Consider whether you can avoid these effects, or mitigate them by performing the onboarding process outside of normal operating hours.

Automation

Automated deployments are always advisable for cloud-hosted solutions. When working with multitenant solutions, deployment automation becomes even more important for the following reasons:

- **Scale:** Manual deployment processes become increasingly complex and time-consuming, as your tenant population increases. An automated deployment approach is easier to scale as the number of tenants grows.
- **Repeatable:** In a multitenant environment, use a consistent process for deployments across all tenants. Manual processes introduce the chance of error, or of steps being performed for some tenants and not others. These manual

processes leave your environment in an inconsistent state, which makes it harder for your team to manage the solution.

- **Impact of outages:** Manual deployments are significantly more risky and prone to outages than automated deployments. In a multitenant environment, the impact of a system-wide outage (due to a deployment error) can be high, since every tenant could be affected.

When you deploy to a multitenant environment, you should use deployment pipelines, and use infrastructure as code (IaC) technologies, such as [Bicep](#), JSON ARM templates, Terraform, or the Azure SDKs.

If you plan to offer your solution through the Azure Marketplace, you should provide a fully automated onboarding process for new tenants. This process is described in the [SaaS fulfillment APIs documentation](#).

Maximum resource capacity

When you programmatically deploy tenant resources onto shared resources, consider the capacity limit for each resource. When you approach that limit, you might need to create another instance of the resource to support further scale. Consider the limits of each resource that you deploy, and the conditions that will trigger you to deploy another instance.

For example, suppose your solution includes an Azure SQL logical server, and your solution provisions a dedicated database on that server for each tenant. A [single logical server has limits](#), which include a maximum number of databases that a logical server supports. As you approach these limits, you might need to provision new servers so that you can continue to onboard tenants. Consider whether you automate this process or manually monitor the growth.

Resource management responsibility

In some multitenant solutions, you deploy dedicated Azure resources for each tenant, such as a database for each tenant. Or, you might decide on a set number of tenants to house on a single instance of a resource, so the number of tenants you have dictates the set of resources that you deploy to Azure. In other solutions, you deploy a single set of shared resources, and you then reconfigure the resources when you onboard new tenants.

Each of these models requires you to deploy and manage resources in different ways, and you must consider how you will deploy and manage the lifecycle of the resources that you provision. Two common approaches are as follows:

- To treat tenants as *configuration* of the resources that you deploy, and use your deployment pipelines to deploy and configure those resources.
- To treat tenants as *data*, and have a [control plane](#) provision and configure infrastructure for your tenants.

Further discussion of these approaches is provided below.

Testing

Plan to thoroughly test your solution during and after every deployment. You can use automated testing to verify the functional and non-functional behavior of your solution. Ensure you test your tenant isolation model, and consider using tools like [Azure Chaos Studio](#) to deliberately introduce faults that simulate real-world outages and verify that your solution functions even when a component is unavailable or malfunctioning.

Approaches and patterns to consider

Several design patterns from the Azure Architecture Center, and the wider community, are of relevance to the deployment and configuration of multitenant solutions.

Deployment Stamps pattern

The [Deployment Stamps pattern](#) involves deploying dedicated infrastructure for a tenant or group of tenants. A single stamp might contain multiple tenants, or it might be dedicated to a single tenant. You can choose to deploy a single stamp, or you can coordinate a deployment across multiple stamps. If you deploy dedicated stamps for each tenant, you can also consider deploying entire stamps programmatically.

Deployment rings

[Deployment rings](#) enable you to roll out updates to different groups of infrastructure at different times. This approach is commonly used with the [Deployment Stamps pattern](#), and groups of stamps can be assigned to distinct rings based on tenant preferences, workload types, and other considerations. For more information, see [Deployment rings](#).

Feature flags

[Feature flags](#) enable you to progressively expose new features or versions of your solution to different tenants, while you maintain a single codebase. Consider using

[Azure App Configuration](#) to manage your feature flags. For more information, see [Feature flags](#).

Sometimes you need to selectively enable specific features for certain customers. For example, you might have different [pricing tiers](#) that allow access to certain capabilities. Feature flags aren't usually the right choice for these scenarios. Instead, consider building a process to track and enforce the *license entitlements* that each customer has.

Antipatterns to avoid

When you deploy and configure multitenant solutions, it's important to avoid situations that inhibit your ability to scale. These include the following:

- **Manual deployment and testing.** As described above, manual deployment processes add risk and slow your ability to deploy. Consider using pipelines for automated deployments, programmatically creating of resources from your solution's code, or a combination of both.
- **Specialized customizations for tenants.** Avoid deploying features or a configuration that only applies to a single tenant. This approach adds complexity to your deployments and testing processes. Instead, use the same resource types and codebase for each tenant, and use strategies like [feature flags](#) for temporary changes or for features that are rolled out progressively, or use [different pricing tiers](#) with license entitlements to selectively enable features for tenants that require them. Use a consistent and automated deployment process, even for tenants that have isolated or dedicated resources.

Tenant lists as configuration or data

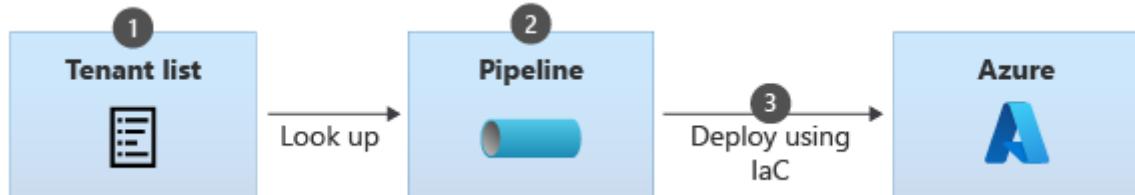
You can consider the following two approaches when you deploy resources in a multitenant solution:

- **Use an automated deployment pipeline to deploy every resource.** As new tenants are added, reconfigure your pipeline to provision the resources for each tenant.
- **Use an automated deployment pipeline to deploy shared resources that don't depend on the number of tenants.** For resources that are deployed for each tenant, create them within your application.

When considering the two approaches, you should distinguish between treating your tenant list as a *configuration* or as *data*. This distinction is also important when you consider how to build a [control plane](#) for your system.

Tenant list as configuration

When you treat your tenant list as configuration, you deploy all your resources from a centralized deployment pipeline. When new tenants are onboarded, you reconfigure the pipeline or its parameters. Typically, the reconfiguration happens through manual changes, as illustrated in the following diagram.



The process to onboard a new tenant might be similar to the following:

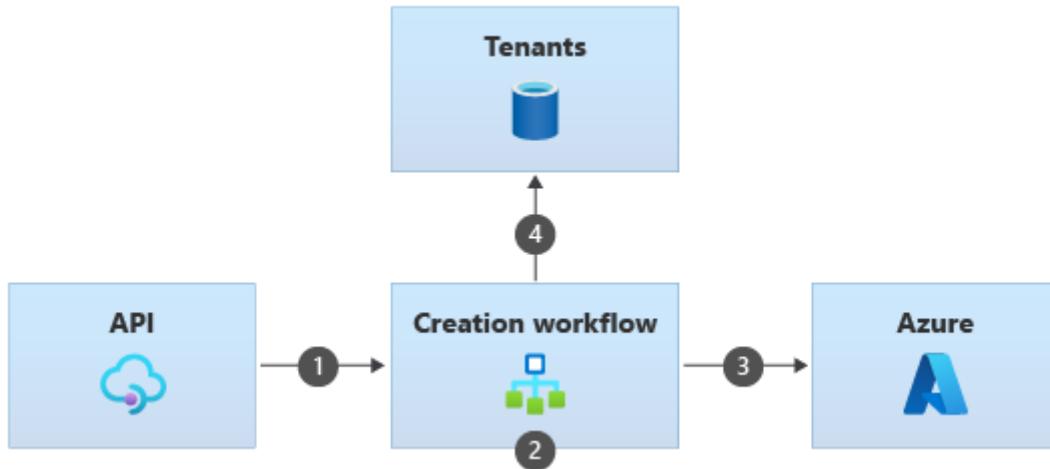
1. Update the tenant list. This typically happens manually by configuring the pipeline itself, or by modifying a parameters file that's included in the pipeline's configuration.
2. Trigger the pipeline to run.
3. The pipeline redeploys your complete set of Azure resources, including any new tenant-specific resources.

This approach tends to work well for small numbers of tenants, and for architectures where all resources are shared. It's a simple approach because all of your Azure resources can be deployed and configured by using a single process.

However, when you approach a larger number of tenants, say 5 to 10 or more, it becomes cumbersome to reconfigure the pipeline as you add tenants. The time it takes to run the deployment pipeline often increases significantly too. This approach also doesn't easily support self-service tenant creation, and the lead time before a tenant is onboarded can be longer because you need to trigger your pipeline to run.

Tenant list as data

When you treat your tenant list as data, you still deploy your shared components by using a pipeline. However, for resources and configuration settings that need to be deployed for each tenant, you imperatively deploy or configure your resources. For example, your control plane can use the [Azure SDKs](#) to create a specific resource or to initiate the deployment of a parameterized template.



The process to onboard a new tenant might be similar to the following, and would happen asynchronously:

1. Request a tenant be onboarded, such as by initiating an API request to your system's control plane.
2. A workflow component receives the creation request and orchestrates the remaining steps.
3. The workflow initiates the deployment of tenant-specific resources to Azure. This could be achieved by using an imperative programming model, such as by using the Azure SDKs, or by imperatively triggering the deployment of a Bicep or Terraform template.
4. When the deployment is completed, the workflow saves the new tenant's details to the central tenant catalog. The data stored for each tenant might include the tenant ID and the resource IDs for all of the tenant-specific resources that the workflow created.

By doing this, you can provision resources for new tenants without redeploying your entire solution. The time involved in provisioning new resources for each tenant is likely to be shorter, because only those resources need to be deployed.

However, this approach is often much more time-consuming to build, and the effort you spend needs to be justified by the number of tenants or the provisioning timeframes you need to meet.

For more information on this approach, see [Considerations for multitenant control planes](#).

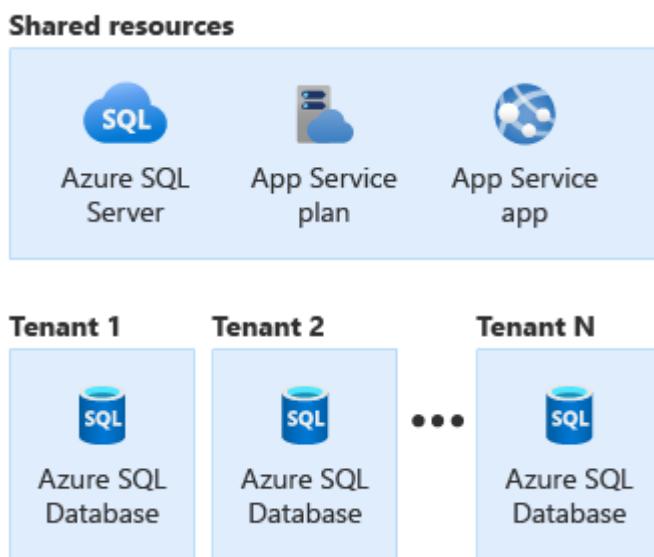
Note

Azure deployment and configuration operations often take time to complete. Ensure you use an appropriate process to initiate and monitor these long-running operations. For example, you might consider following the [Asynchronous Request](#)-

Reply pattern. Use technologies that are designed to support long-running operations, like [Azure Logic Apps](#) and [Durable Functions](#).

Example

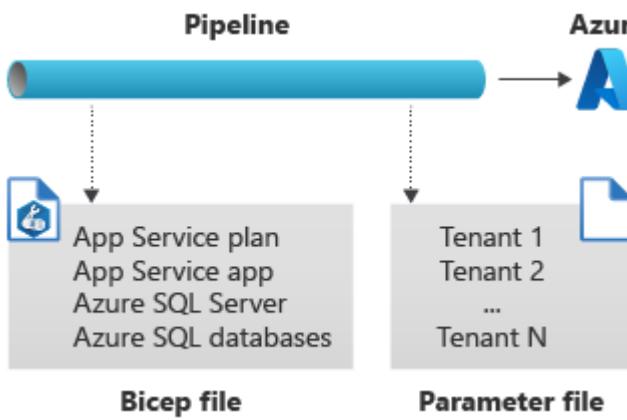
Contoso runs a multitenant solution for their customers. Currently, they have six tenants, and they expect to grow to 300 tenants within the next 18 months. Contoso follows the [Multitenant app with dedicated databases for each tenant](#) approach. They have deployed a single set of App Service resources and an Azure SQL logical server that are shared between all of their tenants, and they deploy a dedicated Azure SQL database for each tenant, as shown in the following diagram.



Contoso uses Bicep to deploy their Azure resources.

Option 1 - Use deployment pipelines for everything

Contoso might consider deploying all of their resources by using a deployment pipeline. Their pipeline deploys a Bicep file that includes all of their Azure resources, including the Azure SQL databases for each tenant. A parameter file defines the list of tenants, and the Bicep file uses a [resource loop](#) to deploy a database for each of the listed tenants, as shown in the following diagram.

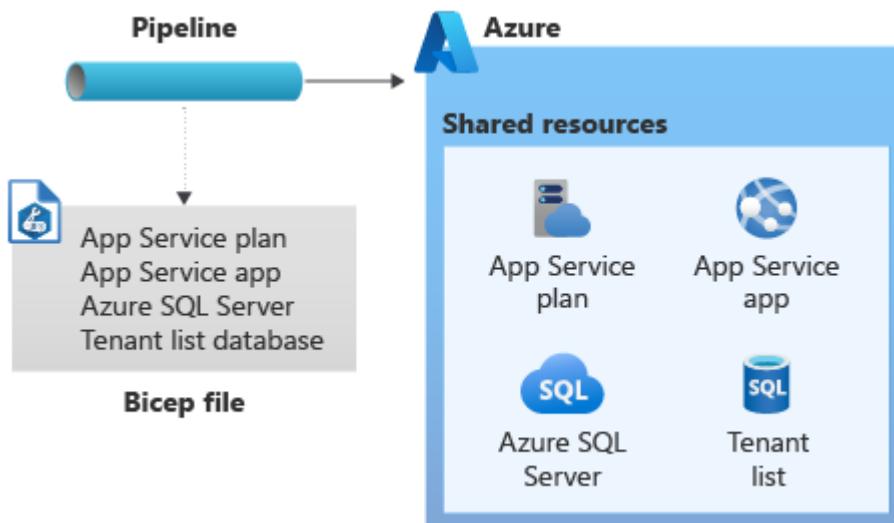


If Contoso follows this model, then they need to update their parameter file as part of the onboarding of a new tenant. Then they need to re-run their pipeline. Also, they need to manually keep track of whether they are near any limits, such as if they grow at an unexpectedly high rate and approach the maximum number of databases that are supported on a single Azure SQL logical server.

Option 2 - Use a combination of deployment pipelines and imperative resource creation

Alternatively, Contoso might consider separating the responsibility for the Azure deployments.

Contoso uses a Bicep file that defines the shared resources that should be deployed. The shared resources support all of their tenants and include a tenant map database, as shown in the following diagram.

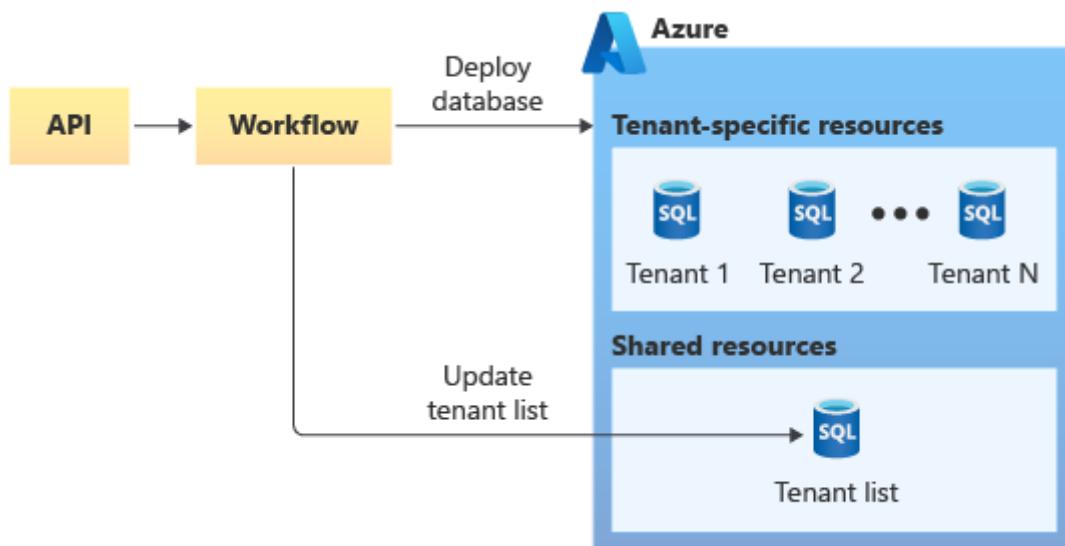


The Contoso team then builds a control plane that includes a tenant onboarding API. When their sales team has completed the sale to a new customer, Microsoft Dynamics triggers the API to begin the onboarding process. Contoso also provides a self-service web interface for customers to use, and that triggers the API too.

The API asynchronously starts a workflow that onboard their new tenants. The workflow initiates the deployment of a new Azure SQL database, which might be done by one of the following approaches:

- Use the Azure SDK to initiate the deployment of a second Bicep file that defines the Azure SQL database.
- Use the Azure SDK to imperatively create an Azure SQL database, by using the [management library](#).

After the database is deployed, the workflow adds the tenant to the tenant list database, as shown in the following diagram.



Ongoing database schema updates are initiated by their application tier.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Review the [considerations for updating a multitenant solution](#).
- Consider [architectural approaches for storage and data](#).
- Consider [how to use Azure Resource Manager in a multitenant solution](#).

Architectural approaches for compute in multitenant solutions

Article • 03/24/2023

Most cloud-based solutions are composed of compute resources of some kind, such as web and application tiers, batch processors, scheduled jobs, and even specialized resources like GPUs and high-performance compute (HPC). Multitenant solutions often benefit from shared compute resources, because a higher density of tenants to infrastructure reduces the operational cost and management. You should consider the isolation requirements and the implications of shared infrastructure.

This article provides guidance about the considerations and requirements that are essential for solution architects to consider when planning a multitenant compute tier. This includes some common patterns for applying multitenancy to compute services, along with some antipatterns to avoid.

Key considerations and requirements

Multitenancy, and the isolation model you select, impacts the scaling, performance, state management, and security of your compute resources. In this section, we review some of the key decisions you must make when you plan a multitenant compute solution.

Scale

Systems need to perform adequately under changing demand. As the number of tenants and the amount of traffic increase, you might need to increase the capacity of your resources, to keep up with the growing number of tenants and to maintain an acceptable performance rate. Similarly, when the number of active users or the amount of traffic decrease, you should automatically reduce the compute capacity to reduce costs, but you should reduce the capacity with minimal impact to users.

If you deploy dedicated resources for each tenant, you have the flexibility to scale each tenant's resources independently. In a solution where compute resources are shared between multiple tenants, if you scale those resources, then all of those tenants can make use of the new scale. However, they also will all suffer when the scale is insufficient to handle their overall load. For more information, see the [Noisy Neighbor problem](#).

When you build cloud solutions, you can choose whether to [scale horizontally or vertically](#). In a multitenant solution with a growing number of tenants, scaling horizontally typically provides you with greater flexibility and a higher overall scale ceiling.

Performance problems often remain undetected until an application is under load. You can use a fully managed service, such as [Azure Load Testing](#), to learn how your application behaves under stress.

Scale triggers

Whichever approach you use to scale, you typically need to plan the triggers that cause your components to scale. When you have shared components, consider the workload patterns of every tenant who uses the resources, in order to ensure your provisioned capacity can meet the total required capacity, and to minimize the chance of a tenant experiencing the [Noisy Neighbor problem](#). You might also be able to plan your scaling capacity by considering the number of tenants. For example, if you measure the resources that you use to service 100 tenants, then as you onboard more tenants, you can plan to scale such that your resources double for every additional 100 tenants.

State

Compute resources can be *stateless*, or they can be *stateful*. Stateless components don't maintain any data between requests. From a scalability perspective, stateless components are often easy to scale out because you can quickly add new workers, instances, or nodes, and they can immediately start to process requests. If your architecture allows for it, you can also repurpose instances that are assigned to one tenant and allocate them to another tenant.

Stateful resources can be further subdivided, based on the type of state they maintain. *Persistent state* is data that needs to be permanently stored. In cloud solutions, you should avoid storing a persistent state in your compute tier. Instead, use storage services like databases or storage accounts. *Transient state* is data that is stored temporarily, and it includes read-only in-memory caches, and the storage of temporary files on local disks.

Transient state is often useful to improve the performance of your application tier, by reducing the number of requests to backend storage services. For example, when you use an in-memory cache, you might be able to serve read requests, without connecting to a database, and without performing an intensive query that you recently performed when you served another request.

In latency-sensitive applications, the cost of cache hydration can become significant. A multitenant solution can exacerbate this issue, if each tenant requires different data to be cached. To mitigate this issue, some solutions use *session affinity* to ensure that all requests for a specific user or tenant are processed by the same compute worker node. Although session affinity can improve the ability of the application tier to use its cache effectively, it also makes it harder to scale and to balance the traffic load across workers. This tradeoff needs to be carefully considered. For many applications, session affinity is not required.

It's also possible to store data in external caches, such as Azure Cache for Redis. External caches are optimized for low-latency data retrieval, while keeping the state isolated from the compute resources, so they can be scaled and managed separately. In many solutions, external caches enable you to improve application performance, while you keep the compute tier stateless.

Important

Avoid leaking data between tenants, whenever you use in-memory caches or other components that maintain state. For example, consider prepending a tenant identifier to all cache keys, to ensure that data is separated for each tenant.

Isolation

When you design a multitenant compute tier, you often have many options to consider for the level of isolation between tenants, including deploying [shared compute resources](#), to be used by all tenants, [dedicated compute resources](#) for each tenant, or [something in between these extremes](#). Each option comes with tradeoffs. To help you decide which option suits your solution best, consider your requirements for isolation.

You might be concerned with the logical isolation of tenants, and how to separate the management responsibilities or policies that are applied to each tenant. Alternatively, you might need to deploy distinct resource configurations for specific tenants, such as deploying a specific virtual machine SKU to suit a tenant's workload.

Whichever isolation model you select, ensure you verify your tenant data remains appropriately isolated even when components are unavailable or malfunctioning. Consider using [Azure Chaos Studio](#) as part of your regular automated testing process to deliberately introduce faults that simulate real-world outages and verify that your solution doesn't leak data between tenants and is functioning properly even under pressure.

Approaches and patterns to consider

Autoscale

Azure compute services provide different capabilities to scale your workloads. [Many compute services support autoscaling](#), which requires you to consider when you should scale, and your minimum and maximum levels of scale. The specific options available for scaling depend on the compute services you use. See the following example services:

- **Azure App Service:** [Specify autoscale rules](#) that scale your infrastructure, based on your requirements.
- **Azure Functions:** Select from [multiple scale options](#), including an event-driven scaling model that automatically scales, based on the work that your functions perform.
- **Azure Container Apps:** Use [event-driven autoscaling](#) to scale your application, based on the work it performs and on its current load.
- **Azure Kubernetes Service (AKS):** To keep up with your application's demands, you might need to [adjust the number of nodes that run your workloads](#). Additionally, to rapidly scale application workloads in an AKS cluster, you can use [virtual nodes](#).
- **Virtual machines:** A virtual machine scale set can [automatically increase or decrease the number of VM instances](#) that run your application.

Deployment Stamps pattern

For more information about how the [Deployment Stamps pattern](#) can be used to support a multitenant solution, see [Overview](#).

Compute Resource Consolidation pattern

The [Compute Resource Consolidation pattern](#) helps you achieve a higher density of tenants to compute infrastructure, by sharing the underlying compute resources. By sharing compute resources, you are often able to reduce the direct cost of those resources. Additionally, your management costs are often lower because there are fewer components to manage.

However, compute resource consolidation increases the likelihood of the [Noisy Neighbor problem](#). Any tenant's workload might consume a disproportionate amount of the compute capacity that's available. You can often mitigate this risk by ensuring you scale your solution appropriately, and by applying controls like quotas and API limits, to avoid tenants that consume more than their fair share of the capacity.

This pattern is achieved in different ways, depending on the compute service you use. See the following example services:

- **Azure App Service and Azure Functions:** Deploy shared App Service plans, which represent the hosting server infrastructure.
- **Azure Container Apps:** Deploy shared environments.
- **Azure Kubernetes Service (AKS):** Deploy shared pods, with a multitenancy-aware application.
- **Virtual machines:** Deploy a single set of virtual machines for all tenants to use.

Dedicated compute resources per tenant

You can also deploy dedicated compute resources for every tenant. Dedicated resources mitigate the risk of the [Noisy Neighbor problem](#), by ensuring that the compute resources for every tenant are isolated from the others. It also enables you to deploy a distinct configuration for each tenant's resources, based on their requirements. However, dedicated resources typically come with a higher cost, because you have a lower density of tenants to resources.

Depending on the Azure compute services you use, you need to deploy different dedicated resources, as follows:

- **Azure App Service and Azure Functions:** Deploy separate App Service plans for each tenant.
- **Azure Container Apps:** Deploy separate environments for each tenant.
- **Azure Kubernetes Service (AKS):** Deploy dedicated clusters for each tenant.
- **Virtual machines:** Deploy dedicated virtual machines for each tenant.

Semi-isolated compute resources

Semi-isolated approaches require you to deploy aspects of the solution in an isolated configuration, while you share the other components.

When you work with App Service and Azure Functions, you can deploy distinct applications for each tenant, and you can host the applications on shared App Service plans. This approach reduces the cost of your compute tier, because App Service plans represent the unit of billing. It also enables you to apply distinct configuration and policies to each application. However, this approach introduces the risk of the [Noisy Neighbor problem](#).

Azure Container Apps enables you to deploy multiple applications to a shared environment, and then to use Dapr and other tools to configure each application

separately.

Azure Kubernetes Service (AKS), and Kubernetes more broadly, provide a variety of options for multitenancy, including the following:

- Tenant-specific namespaces, for logical isolation of tenant-specific resources, which are deployed to shared clusters and node pools.
- Tenant-specific nodes or node pools on a shared cluster.
- Tenant-specific pods that might use the same node pool.

AKS also enables you to apply pod-level governance to mitigate the [Noisy Neighbor problem](#). For more information, see [Best practices for application developers to manage resources in Azure Kubernetes Service \(AKS\)](#).

It's also important to be aware of shared components in a Kubernetes cluster, and how these components might be affected by multitenancy. For example, the Kubernetes API server is a shared service that is used throughout the entire cluster. Even if you provide tenant-specific node pools to isolate the tenants' application workloads, the API server might experience contention from a large number of requests across the tenants.

Antipatterns to avoid

Noisy Neighbor antipattern

Whenever you deploy components that are shared between tenants, the [Noisy Neighbor problem](#) is a potential risk. Ensure you include resource governance and monitoring to mitigate the risk of a tenant's compute workload being affected by the activity of other tenants.

Cross-tenant data leakage

Compute tiers can be subject to cross-tenant data leakage, if they are not properly handled. This isn't generally something you need to consider when you're using a multitenant service on Azure, because Microsoft provides protections at the platform layer. However, when you develop your own multitenant application, consider whether any shared resources (such as local disk caches, RAM, and external caches) might contain data that another tenant can inadvertently view or modify.

Busy Front End antipattern

To avoid the [Busy Front End antipattern](#), avoid your front end tier doing a lot of the work that could be handled by other components or tiers of your architecture. This antipattern is particularly important when you create shared front-ends for a multitenant solution, because a busy front end will degrade the experience for all tenants.

Instead, consider using asynchronous processing by making use of queues or other messaging services. This approach also enables you to apply *quality of service* (QoS) controls for different tenants, based on their requirements. For example, all tenants might share a common front end tier, but tenants who [pay for a higher service level](#) might have a higher set of dedicated resources to process the work from their queue messages.

Inelastic or insufficient scaling

Multitenant solutions are often subject to bursty scale patterns. Shared components are particularly susceptible to this issue, because the scope for burst is higher, and the impact is greater when you have more tenants with distinct usage patterns.

Ensure you make good use of the elasticity and scale of the cloud. Consider whether you should use [horizontal or vertical scaling](#), and use autoscaling to automatically handle spikes in load. Test your solution to understand how it behaves under different levels of load. Ensure you include the load volumes that are expected in production, and your expected growth. You can use a fully managed service, such as [Azure Load Testing](#), to learn how your application behaves under stress.

No Caching antipattern

The [No Caching antipattern](#) is when the performance of your solution suffers because the application tier repeatedly requests or recomputes information that could be reused across requests. If you have data that can be shared, either among tenants or among users within a single tenant, it's likely worth caching it to reduce the load on your backend/database tier.

Unnecessary statefulness

The corollary to the No Caching antipattern is that you also should avoid storing unnecessary state in your compute tier. Be explicit about where you maintain state and why. Stateful front-end or application tiers can reduce your ability to scale. Stateful compute tiers typically also require session affinity, which can reduce your ability to effectively load balance traffic, across workers or nodes.

Consider the tradeoffs for each piece of state you maintain in your compute tier, and whether it impacts your ability to scale or to grow as your tenants' workload patterns change. You can also store state in an external cache, such as Azure Cache for Redis.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- Dixit Arora | Senior Customer Engineer, FastTrack for Azure
- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure

Next steps

Review service-specific guidance for your compute services:

- [Azure App Service and Azure Functions considerations for multitenancy](#)
- [Considerations for using Container Apps in a multitenant solution](#)
- [Azure Kubernetes Service \(AKS\) considerations for multitenancy](#)

Architectural approaches for control planes in multitenant solutions

Article • 11/07/2023

Control planes are an important part of software as a service (SaaS) and multitenant solutions, especially to help manage a solution at scale. Typically, there are two main components that make up a control plane:

- The tenant catalog, which stores important information about your tenants, such as:
 - Tenant configuration.
 - SKUs deployed for tenant resources.
 - Which [deployment stamps](#) the tenants are allocated to.
- Processes for managing changes to the environment, which are triggered by [tenant lifecycle events](#). For example, tenant onboarding, tenant offboarding, and any required regular maintenance.

A control plane is itself an application. You need to think about your control plane carefully and design it with the same rigor and care you use with any other part of your solution. For more information on what a control plane is, why you should use it, and considerations for designing one, see [Considerations for multitenant control planes](#).

This article describes some approaches you can consider for designing and creating a control plane. The list of approaches described here isn't comprehensive. Although the approaches are all valid, there are other valid architectures.

Approaches and patterns to consider

The following table summarizes the differences between some of the approaches you can consider for a control plane. Manual, low-code, and custom approaches are compared.

Consideration	Manual	Low-code	Custom
Operational overhead	High	Low-medium	Low
Frequency of lifecycle events the approach is suitable for	Rare	Occasional-often	Often
Time and complexity to implement	Low	Medium	High
Control plane maintenance responsibilities	Low	Medium	High

Consideration	Manual	Low-code	Custom
Testability	Low	Medium	High
Risk of inconsistencies	High	Medium-low	Low

Manual processes

It's not always essential to build a fully automated control plane, especially when you're starting out and have only a small number of tenants.

You might keep your tenant catalog somewhere centrally located, like in an Excel workbook or a JSON file that's stored in a place that your team can access. Regardless of the format, it's a good idea to store the information in a structured way so that you can easily work with the data programmatically.

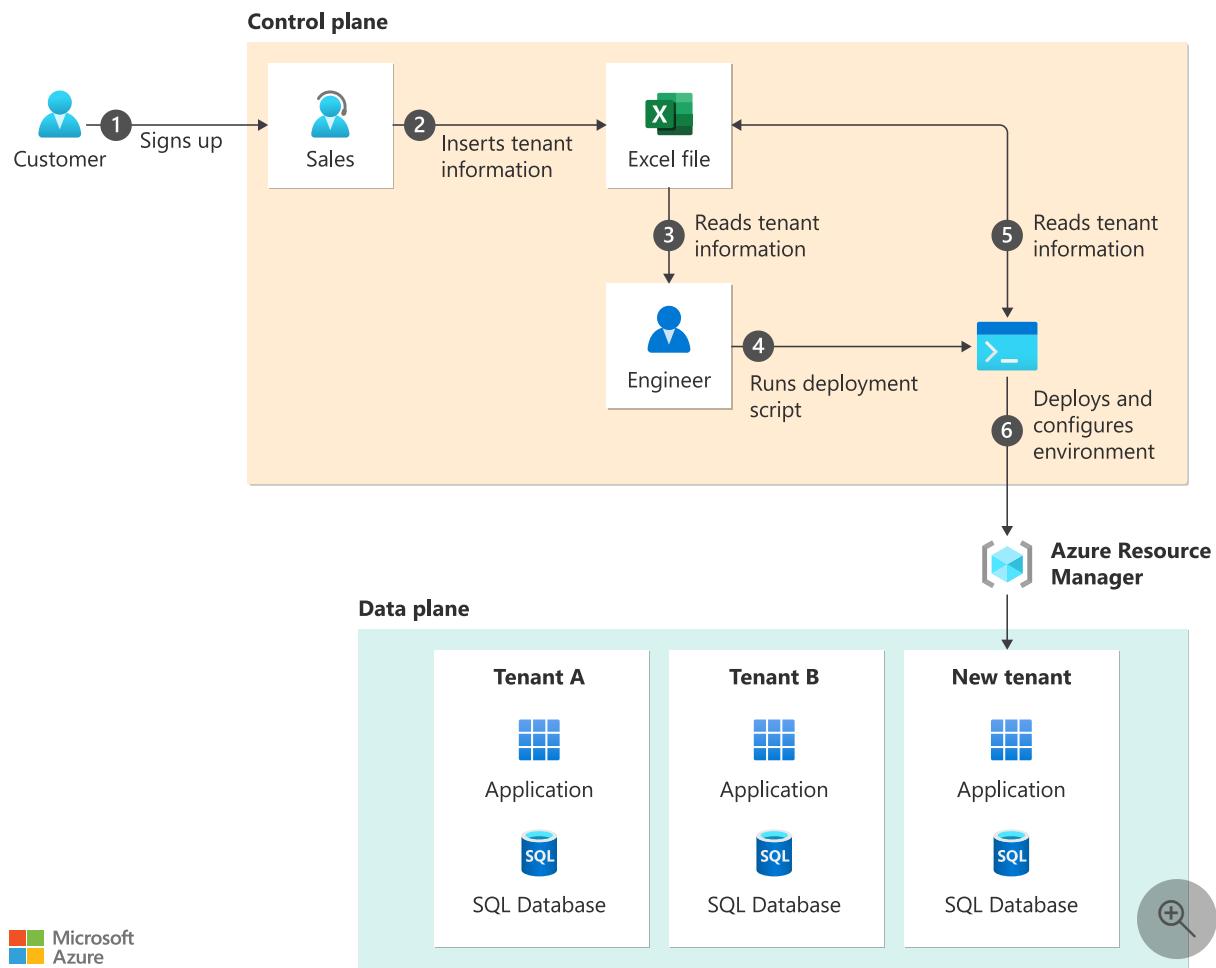
ⓘ Note

A manual control plane is a great way to get started with managing your multitenant application, but it's only suitable for a small number of tenants (less than 5-10). The administrative overhead and the risk of inconsistencies increase with each tenant you onboard manually. You should only use this approach if you have only a few tenants and you don't need automated or self-service onboarding.

For processes like tenant onboarding and maintenance activities:

- ✓ **Create scripts or automated pipelines wherever possible, even if you run them manually.** By using scripts or pipelines, you ensure that the steps run consistently for each tenant.
- ✓ **For tasks that you can't script initially, document the process thoroughly and in explicit detail.** Document the *how* as well as the *why*. If somebody ends up automating the task in the future, they should have a good understanding of both.

The following diagram illustrates one way to use manual processes for an initial control plane:



Download a [Visio file](#) of this architecture.

Advantages of a manual approach

- **Lightweight:** Documentation, scripts, and pipelines are easy to develop and modify. This makes them appropriate when you're figuring out your processes because you can rapidly iterate and evolve them.
- **Low cost:** Maintaining and running a manual approach is inexpensive.
- **Validates your process:** Even if you eventually intend to use a more automated approach, starting with a manual approach as a proof of concept is a good way to validate your maintenance strategy before you invest time in developing more robust automation.

Disadvantages of a manual approach

- **Lack of control:** This approach relies on everybody involved doing the correct thing. Somebody might deviate from the prescribed processes, either accidentally or intentionally. Every variation in process increases the risk of inconsistency in your environment, which makes ongoing management much harder.

- **Access-control challenges:** When you use this approach, you typically need to grant broadly scoped, highly permissive access to anybody who operates your solution, which makes it hard to follow the best practices for [access segmentation](#).
- **Scalability:** The work required to run manual processes scales with the number of tenants that you need to manage.
- **Testability:** Manual processes are difficult to validate and test.

When to consider moving away from a manual approach

- When your team can't keep up with the amount of work they need to do to maintain the application. For example, when your number of tenants scales beyond a critical point, which for most teams is between 5 and 10 tenants.
- When you anticipate tenant growth beyond a critical number of tenants and you need to prepare for the work involved in administering that number of tenants.
- When you need to mitigate the risk of inconsistencies. For example, you might observe some mistakes occurring because somebody isn't following the processes correctly, or because there's too much ambiguity in the processes. The risk of inconsistency typically grows as more tenants are onboarded manually.

Low-code control plane

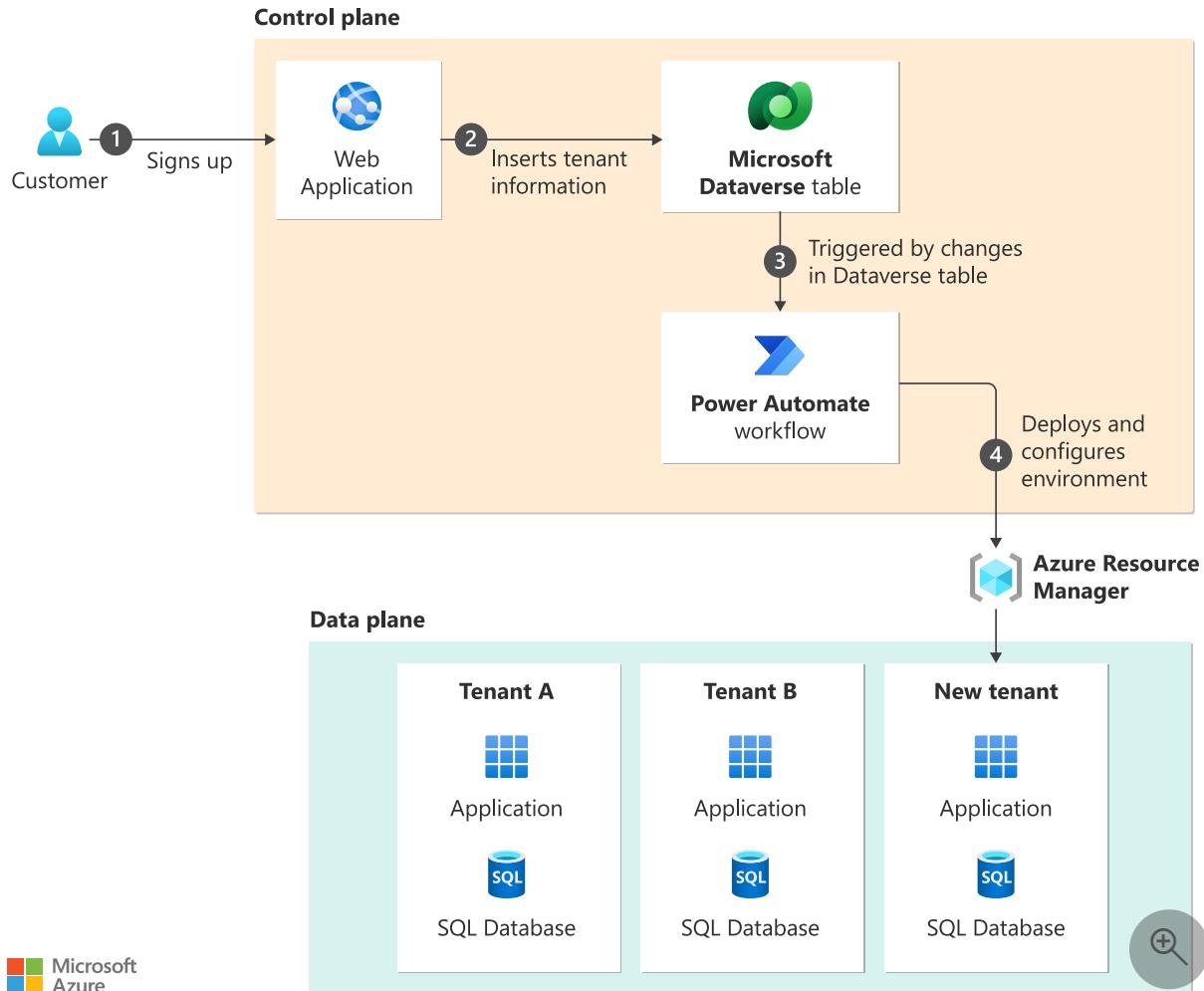
A low-code or no-code control plane is a type of control plane that's built on a platform that's designed to enable you to automate business processes and track information. There are many platforms that enable you to do that without writing custom code. Using one of these platforms can be a good approach to the low-code creation of a control plane.

Microsoft Power Platform is an example of one of these platforms. If you use Power Platform, you might keep your tenant catalog in Dynamics 365, Dataverse, or Microsoft 365. You can also consider keeping the same tenant catalog that you use for your manual processes, if you don't want to fully commit yet to automating everything.

For tenant onboarding and maintenance, you could then use Power Automate to run workflows that perform tenant management, configure tenants, trigger pipelines or API calls, and so on. You could use Power Automate to watch for changes to your tenant catalog, if the data is somewhere accessible to Power Automate. If you use a manual tenant catalog, Power Automate workflows can also be triggered manually. You might decide to include manual approval steps in your workflows if you need somebody from your team to verify something or perform additional steps that can't be fully automated.

This approach enables you to provide self-service sign-up to your customers by allowing your web application to directly add records to your tenant catalog without human intervention.

The following diagram illustrates how you might create a control plane with self-service sign-up by using the Microsoft Power Platform:



[Download a Visio file](#) of this architecture.

Advantages of a low-code approach

- **Lightweight:** It's often quick and inexpensive to create a set of low-code workflows and connect it to the surrounding systems.
- **Uses platform tooling:** You can use native platform features to store data, create administrative portals for your team to use, and monitor the workflows as they run.
- **Customizable:** If you need more customization, you can typically augment your workflows with custom code and processes. For example, you might use Power Automate to trigger a deployment workflow in GitHub Actions, or you might invoke Azure Functions to run your own code. This also helps facilitate a gradual implementation.

- **Low overhead:** Low-code services are typically fully managed, so you don't need to manage infrastructure.

Disadvantages of a low-code approach

- **Required expertise:** To use low-code platforms to create processes, and to effectively use these platforms, you typically need proprietary knowledge. Many organizations already use these tools, so your team might already have the required expertise, but it might not. You should consider whether you need to train your team in order to effectively use these platforms.
- **Management:** It can be challenging to handle the management of large amounts of low-code configuration.
- **Testability:** Consider how to test and promote changes to your control plane. In a managed platform, creating a typical DevOps process for testing and promoting changes is more difficult, because changes are normally made through configuration, not through code.
- **Design:** Think carefully about how to meet non-functional requirements like security and reliability. These requirements are often managed for you on a low-code platform.

When to consider moving away from a low-code approach

- Eventually, your requirements might become so complex that you can't sensibly incorporate them in a low-code solution. When you need to work around tooling limitations to meet your needs, it probably makes sense to move away from a managed solution and toward a custom control plane.

Custom control plane

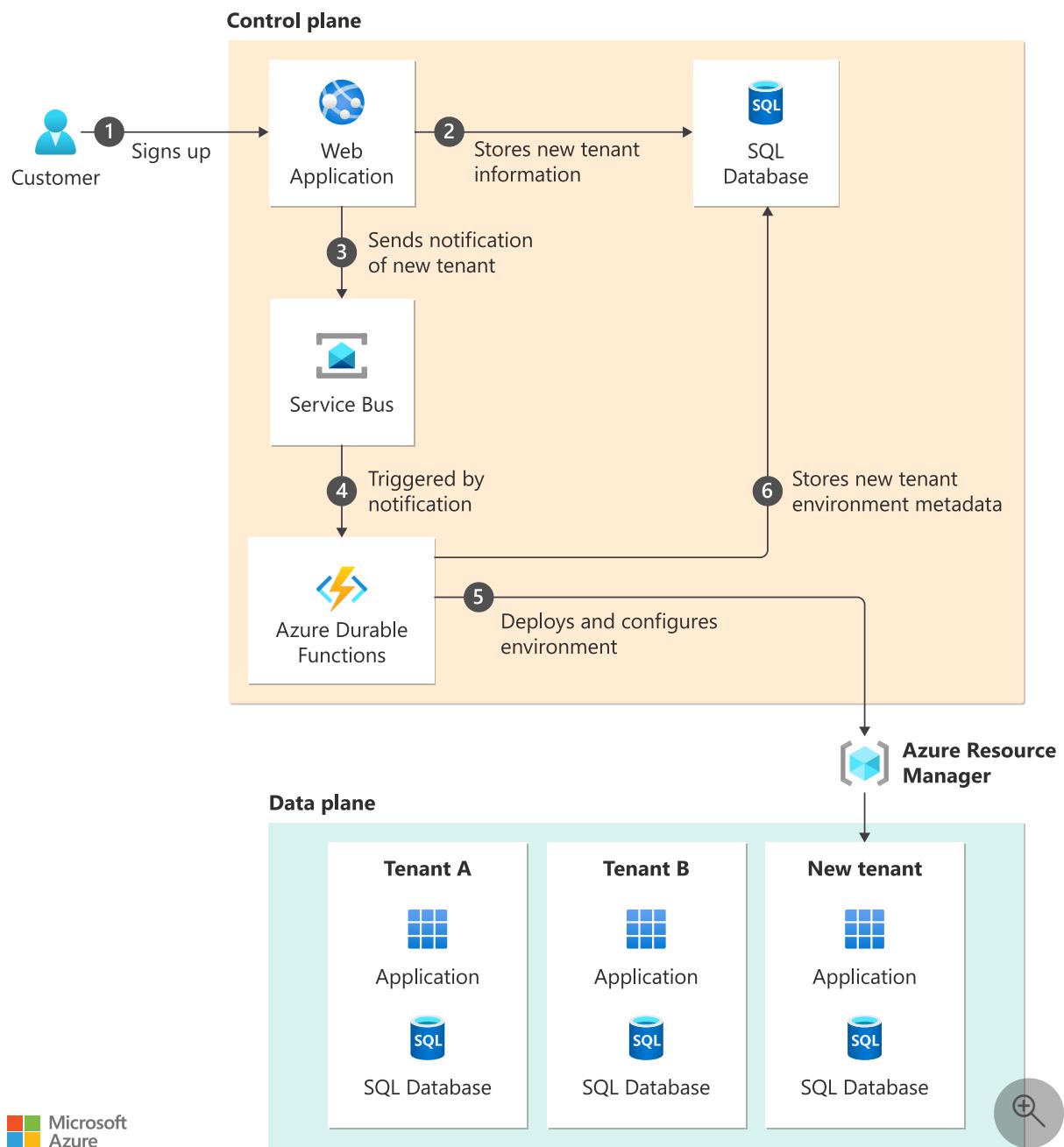
You can also consider creating your own completely customized control plane. This option provides the most flexibility and power, but it also requires the most work. The tenant catalog is usually stored in a database. You don't work directly with the catalog in this case, but instead manage it through an administrative interface, which might be a custom application or a system like your organization's customer relationship management (CRM) application.

You typically create a set of control plane components that's designed around all your tenant administrative functions. These components might include an administrative portal or other user interface, an API, and background processing components. If you need to do things like deploy code or infrastructure when tenant lifecycle events occur, deployment pipelines might also make up your control plane.

Ensure that any long-running processing uses appropriate tooling. For example, you might use [Durable Functions](#) or [Azure Logic Apps](#) for components that orchestrate tenant onboarding or deployments, or for components that need to communicate with external systems.

Like the low-code approach, this approach enables you to provide self-service sign-up to your customers by allowing your web application to directly add records to your tenant catalog without human intervention.

The following diagram shows one way to create a basic custom control plane that provides self-service sign-up:



[Download a Visio file](#) of this architecture.

Advantages of a custom approach

- **Full flexibility and customizability:** You have complete control over what your control plane does and can change it if your requirements change.
- **Testability:** You can use a standard software development lifecycle (SDLC) for your control plane application and implement normal approaches for testing and deployments, just like you would for your main applications.

Disadvantages of a custom approach

- **Maintenance responsibilities:** This approach requires more maintenance overhead because you need to create everything yourself. A control plane is as important as any other part of your application. You need to take great care in developing and running your control plane to ensure it's reliable and secure.

Hybrid approaches

You can also consider using a hybrid approach. You might use a combination of manual and automated systems, or you might use a managed platform like Microsoft Power Platform and augment it with custom applications. Consider implementing a hybrid approach if you need the customizability of the custom control plane but don't necessarily want to build and maintain a fully custom system. Keep in mind that, at some point, your automated customizations to your manual processes or your managed platform might become as complex as a fully customized system. The tipping point is different for every organization, but if your hybrid approach is cumbersome to maintain, you should consider building a fully custom system.

Gradual implementation

Even if you know that you want to eventually automate your control plane, you don't necessarily need to start with that approach. A common approach during the initial stages of creating your application is to start with a manual control plane. As your application progresses and onboards more tenants, you should begin to identify bottleneck areas and automate them as necessary, moving to a hybrid approach. As you automate more, you might eventually have a fully automated control plane.

Antipatterns to avoid

- **Relying on manual processes for too long.** Although it's reasonable to use manual processes when you start out or when you have a low number of tenants and

require fairly lightweight management, you need to plan how to scale to an automated solution as you grow. If you need to hire additional resources to keep up with the demand of your manual processes, that's a good sign that you should start automating parts of your control plane.

- **Using inappropriate tools for long-running workflows.** For example, avoid using standard Azure functions, synchronous API calls, or other tools that have an execution time limit to perform long-running operations like Azure Resource Manager deployments or multi-step orchestrations. For more information, see [Azure Functions performance and reliability](#) and [Asynchronous Request-Reply pattern](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Program Manager
- [Landon Pierce](#) | Customer Engineer

Other contributors:

- [Mick Alberts](#) | Technical Writer
- [Bohdan Cherchyk](#) | Senior Customer Engineer
- [Arsen Vladimirskiy](#) | Principal Customer Engineer

Next steps

- [Microsoft Power Platform](#)
- [Power Automate](#)

Related resources

- [Architectural considerations for control planes in a multitenant solution](#)
- [Architectural approaches for a multitenant solution](#)

Architectural approaches for networking in multitenant solutions

Article • 03/24/2023

All solutions deployed to Azure require networking of some kind. Depending on your solution design and the workload, the ways in which you interact with Azure's networking services might be different. In this article, we provide considerations and guidance for the networking aspects of multitenant solutions on Azure. We include information about the lower-level networking components, like virtual networks, through to higher-level and application-tier approaches.

ⓘ Note

Azure itself is a multitenant environment, and Azure's network components are designed for multitenancy. Although it's not required to understand the details in order to design your own solution, you can [learn more about how Azure isolates your virtual network traffic from other customers' traffic](#).

Key considerations and requirements

Infrastructure and platform services

The concerns you have for your networking components will differ, depending on the category of services you use.

Infrastructure services

Whenever you use infrastructure services, like virtual machines or Azure Kubernetes Service, you need to consider and design the virtual networks, or VNets, that underpin your services' connectivity. You also need to consider the other layers of security and isolation that you need to incorporate in your design. [Avoid relying exclusively on network-layer controls](#).

Platform services

If you use Azure's platform services, like App Service, Azure Cosmos DB, or Azure SQL Database, then the specific architecture you use will determine the networking services

you require.

If you need to isolate your platform services from the internet, you need to use a VNet. Depending on the specific services you use, you might work with [private endpoints](#) or VNet-integrated resources, like [Application Gateway](#). However, you might also choose to make your platform services available through their public IP addresses, and use the services' own protections like firewalls and identity controls. In these situations, you might not need a VNet.

The decision of whether to use VNets for platform services is based on many requirements, including the following factors:

- **Compliance requirements:** You might need to meet a specific compliance standard. Some standards require your Azure environment to be configured in specific ways.
- **Your tenants' requirements:** Even if your own organization doesn't have specific requirements for network-layer isolation or controls, your tenants might. Ensure you have a clear understanding of how your tenants will access your solution and whether they have any assumptions about its network design.
- **Complexity:** It can be more complex to understand and work with virtual networks. Ensure your team has a clear understanding of the principles involved, or you're likely to deploy an insecure environment.

Ensure that you understand the [implications of using private networking](#).

Sizing subnets

When you need to deploy a VNet, it's important to carefully consider the sizing and address space of the entire VNet and of the subnets within the VNet.

Ensure you have a clear understanding of how you will deploy your Azure resources into VNets, and the number of IP addresses each resource consumes. If you deploy tenant-specific compute nodes, database servers, or other resources, ensure you create subnets that are large enough for your expected tenant growth and [horizontal autoscaling of resources](#).

Similarly, when you work with managed services, it's important that you understand how IP addresses are consumed. For example, when you use Azure Kubernetes Service with [Azure Container Networking Interface \(Azure CNI\)](#), the number of IP addresses consumed from a subnet will be based on factors like the number of nodes, how you scale horizontally, and the service deployment process that you use. When you use Azure App Service and Azure Functions with VNet integration, [the number of IP addresses consumed is based on the number of plan instances](#).

Review the subnet segmentation guidance when planning your subnets.

Public or private access

Consider whether your tenants will access your services through the internet or through private IP addresses.

For internet-based (public) access, you can use firewall rules, IP address allowlisting and denylisting, shared secrets and keys, and identity-based controls to secure your service.

If you need to enable connectivity to your service by using private IP addresses, consider using [Azure Private Link Service](#) or [cross-tenant virtual network peering](#). For some limited scenarios, you might also consider using Azure ExpressRoute or Azure VPN Gateway to enable private access to your solution. We only advise that you use this approach when you have a small number of tenants, and deploy dedicated VNets for each tenant.

Access to tenants' endpoints

Consider whether you need to send data to endpoints within the tenants' networks, either within or outside of Azure. For example, will you need to invoke a webhook that's provided by a customer, or do you need to send real-time messages to a tenant?

If you do need to send data to tenants' endpoints, consider the following common approaches:

- Initiate connections from your solution to tenants' endpoints through the internet. Consider whether the connections must originate from a [static IP address](#). Depending on the Azure services you use, you might need to deploy a [NAT Gateway](#), firewall, or load balancer.
- Deploy an [agent](#) to enable connectivity between your Azure-hosted services and your customers' networks, regardless of where they are located.
- For one-way messaging, consider using a service like [Azure Event Grid](#), with or without [event domains](#).

Approaches and patterns to consider

In this section, we describe some of the key networking approaches that you can consider in a multitenant solution. We begin by describing the lower-level approaches for core networking components, and then follow with the approaches that you can consider for HTTP and other application-layer concerns.

Tenant-specific VNets with service provider-selected IP addresses

In some situations, you need to run dedicated VNet-connected resources in Azure on a tenant's behalf. For example, you might run a virtual machine for each tenant, or you might need to use private endpoints to access tenant-specific databases.

Consider deploying a VNet for each tenant, by using an IP address space that you control. This approach enables you to peer the VNets together for your own purposes, such as if you need to establish a [hub and spoke topology](#) to centrally control traffic ingress and egress.

However, service provider-selected IP addresses aren't appropriate if tenants need to connect directly to the VNet you created, such as by using VNet peering. It's likely that the address space you select will be incompatible with their own address spaces.

Tenant-specific VNets with tenant-selected IP addresses

If tenants need to peer their own VNets with the VNet you manage on their behalf, consider deploying tenant-specific VNets with an IP address space that the tenant selects. This system enables each tenant to ensure that the IP address ranges in your system's VNet don't overlap with their own VNets. By using non-overlapping IP address ranges, they can ensure their networks are compatible for peering.

However, this approach means it's unlikely that you can peer your tenants' VNets together or adopt a [hub and spoke topology](#), because there are likely to be overlapping IP address ranges among VNets of different tenants.

Hub and spoke topology

The [hub and spoke VNet topology](#) enables you to peer a centralized *hub* VNet with multiple *spoke* VNets. You can centrally control the traffic ingress and egress for your VNets, and control whether the resources in each spoke's VNet can communicate with each other. Each spoke VNet can also access shared components, like Azure Firewall, and it might be able to use services like Azure DDoS Protection.

When you use a hub and spoke topology, ensure you plan around limits, [such as the maximum number of peered VNets](#), and ensure that you don't use overlapping address spaces for each tenant's VNet.

The hub and spoke topology can be useful when you deploy tenant-specific VNets with IP addresses that you select. Each tenant's VNet becomes a spoke, and can share your

common resources in the hub VNet. You can also use the hub and spoke topology when you scale shared resources across multiple VNets for scale purposes, or when you use the [Deployment Stamps pattern](#).

💡 Tip

If your solution runs across multiple geographic regions, it's usually a good practice to deploy separate hubs and hub resources in each region. While this practice incurs a higher resource cost, it avoids traffic going through multiple Azure regions unnecessarily, which can increase the latency of requests and incur global peering charges.

Static IP addresses

Consider whether your tenants need your service to use static public IP addresses for inbound traffic, outbound traffic, or both. Different Azure services enable static IP addresses in different ways.

When you work with virtual machines and other infrastructure components, consider using a load balancer or firewall for both inbound and outbound static IP addressing. Consider using NAT Gateway to control the IP address of outbound traffic. For more information about using NAT Gateway in a multitenant solution, see [Azure NAT Gateway considerations for multitenancy](#).

When you work with platform services, the specific service you use determines whether and how you can control IP addresses. You might need to configure the resource in a specific way, such as by deploying the resource into a VNet and by using a NAT Gateway or firewall. Or, you can request the current set of IP addresses that the service uses for outbound traffic. For example, [App Service provides an API and web interface to obtain the current outbound IP addresses for your application](#).

Agents

If you need to enable your tenants to receive messages that are initiated by your solution, or if you need to access data that exists in tenants' own networks, then consider providing an agent (sometimes called an *on-premises gateway*) that they can deploy within their network. This approach can work whether your tenants' networks are in Azure, in another cloud provider, or on premises.

The agent initiates an outbound connection to an endpoint that you specify and control, and either keeps long-running connections alive or polls intermittently. Consider using

Azure Relay to establish and manage connections from your agent to your service. When the agent establishes the connection, it authenticates and includes some information about the tenant identifier so that your service can map the connection to the correct tenant.

Agents typically simplify the security configuration for your tenants. It can be complex and risky to open inbound ports, especially in an on-premises environment. An agent avoids the need for tenants to take this risk.

Examples of Microsoft services that provide agents for connectivity to tenants' networks include:

- [Azure Data Factory's self-hosted integration runtime](#).
- [Azure App Service Hybrid Connection](#).
- Microsoft on-premises data gateway, which is used for [Azure Logic Apps](#), [Power BI](#), and other services.

Azure Private Link service

[Azure Private Link service](#) provides private connectivity from a tenant's Azure environment to your solution. Tenants can also use Private Link service with their own VNet, to access your service from an on-premises environment. Azure securely routes the traffic to the service using private IP addresses.

For more information about Private Link and multitenancy, see [Multitenancy and Azure Private Link](#).

Domain names, subdomains, and TLS

When you work with domain names and transport-layer security (TLS) in a multitenant solution, there are a number of considerations. [Review the considerations for multitenancy and domain names](#).

Gateway Routing and Gateway Offloading patterns

The [Gateway Routing pattern](#) and the [Gateway Offloading pattern](#) involve deploying a layer 7 reverse proxy or *gateway*. Gateways are useful to provide core services for a multitenant application, including the following capabilities:

- Routing requests to tenant-specific backends or deployment stamps.
- Handling tenant-specific domain names and TLS certificates.

- Inspecting requests for security threats, by using a [web application firewall \(WAF\)](#).
- Caching responses to improve performance.

Azure provides several services that can be used to achieve some or all of these goals, including Azure Front Door, Azure Application Gateway, and Azure API Management. You can also deploy your own custom solution, by using software like NGINX or HAProxy.

If you plan to deploy a gateway for your solution, a good practice is to first build a complete prototype that includes all of the features you need, and to verify that your application components continue to function as you expect. You should also understand how the gateway component will scale to support your traffic and tenant growth.

Static Content Hosting pattern

The [Static Content Hosting pattern](#) involves serving web content from a cloud-native storage service, and using a content delivery network (CDN) to cache the content.

You can use [Azure Front Door](#) or another CDN for your solution's static components, such as single-page JavaScript applications, and for static content like image files and documents.

Depending on how your solution is designed, you might also be able to cache tenant-specific files or data within a CDN, such as JSON-formatted API responses. This practice can help you improve the performance and scalability of your solution, but it's important to consider whether tenant-specific data is isolated sufficiently to avoid leaking data across tenants. Consider how you plan to purge tenant-specific content from your cache, such as when data is updated or a new application version is deployed. By including the tenant identifier in the URL path, you can control whether you purge a specific file, all the files that relate to a specific tenant, or all the files for all the tenants.

Antipatterns to avoid

Failing to plan for VNet connectivity

By deploying resources into VNets, you have a great deal of control over how traffic flows through your solution's components. However, VNet integration also introduces additional complexity, cost, and other factors that you need to consider. This effect is especially true with platform at a service (PaaS) components.

It's important to test and plan your network strategy, so that you uncover any issues before you implement it in a production environment.

Not planning for limits

Azure enforces a number of limits that affect networking resources. These limits include [Azure resource limits](#) and fundamental protocol and platform limits. For example, when you build a high-scale multitenant solution on platform services, such as Azure App Service and Azure Functions, you might need to consider the [number of TCP connections and SNAT ports](#). When you work with virtual machines and load balancers, you also need to consider limitations for [outbound rules](#) and for [SNAT ports](#).

Small subnets

It's important to consider the size of each subnet to allow for the number of resources or instances of resources that you will deploy, especially as you scale. When you work with platform as a service (PaaS) resources, ensure you understand how your resource's configuration and scale will affect the number of IP addresses that are required in its subnet.

Improper network segmentation

If your solution requires virtual networks, consider how you configure [network segmentation](#) to enable you to control inbound and outbound (north-south) traffic flows and the flows within your solution (east-west). Decide whether tenants should have their own VNets, or if you will deploy shared resources in shared VNets. Changing the approach can be difficult, so ensure you consider all of your requirements, and then select an approach that will work for your future growth targets.

Relying only on network-layer security controls

In modern solutions, it's important to combine network-layer security with other security controls, and you should not rely only on firewalls or network segmentation. This is sometimes called *zero-trust networking*. Use identity-based controls to verify the client, caller, or user, at every layer of your solution. Consider using services that enable you to add additional layers of protection. The options you have available depend on the Azure services that you use. In AKS, consider using a service mesh for mutual TLS authentication, and [network policies](#) to control east-west traffic. In App Service, consider using the [built-in support for authentication and authorization](#) and [access restrictions](#).

Rewriting host headers without testing

When you use the [Gateway Offloading pattern](#), you might consider rewriting the `Host` header of HTTP requests. This practice can simplify the configuration of your backend web application service by offloading the custom domain and TLS management to the gateway.

However, `Host` header rewrites can cause problems for some backend services. If your application issues HTTP redirects or cookies, the mismatch in host names can break the application's functionality. In particular, this issue can arise when you use backend services that are themselves multitenant, like Azure App Service, Azure Functions, and Azure Spring Apps. For more information, see the [host name preservation best practice](#).

Ensure you test your application's behavior with the gateway configuration that you plan to use.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [Joshua Waddell](#) | Senior Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Review [considerations when using domain names in a multitenant solution](#).
- Review service-specific guidance for your networking services:
 - [Use Azure Front Door in a multitenant solution](#)
 - [Azure NAT Gateway considerations for multitenancy](#)
 - [Multitenancy and Azure Private Link](#)

Architectural approaches for storage and data in multitenant solutions

Azure Azure SQL Database Azure Storage

Data is often considered the most valuable part of a solution, because it represents your - and your customers' - valuable business information. So, it's important to carefully manage your data. When planning storage or data components for a multitenant system, you need to decide on an approach for sharing or isolating your tenants' data.

In this article, we provide guidance about the key considerations and requirements that are essential for solution architects when deciding on an approach to store data in a multitenant system. We then suggest some common patterns for applying multitenancy to storage and data services, and some antipatterns to avoid. Finally, we provide targeted guidance for some specific situations.

Key considerations and requirements

It's important to consider the approaches you use for storage and data services from a number of perspectives, which approximately align to the pillars of the [Azure Well-Architected Framework](#).

Scale

When you work with services that store your data, you should consider the number of tenants you have, and the volume of data you store. If you have a small number of tenants (such as five or less), and you're storing small amounts of data for each tenant, then it's likely to be a wasted effort to plan a highly scalable data storage approach, or to build a fully automated approach to manage your data resources. But as you grow, you increasingly benefit from having a clear strategy to scale your data and storage resources, and to apply automation to their management. When you have 50 tenants or more, or if you plan to reach that level of scale, then it's especially important to design your data and storage approach, with scale as a key consideration.

Consider the extent to which you plan to scale, and clearly plan your data storage architectural approach to meet that level of scale.

Performance predictability

Multitenant data and storage services are particularly susceptible to the [Noisy Neighbor problem](#). It's important to consider whether your tenants might affect each other's performance. For example, do your tenants have overlapping peaks in their usage patterns over time? Do all of your customers use your solution at the same time each day, or are requests distributed evenly? Those factors will impact the level of isolation you need to design for, the amount of resources you need to provision, and the degree to which resources can be shared between tenants.

It's important to consider [Azure's resource and request quotas](#) as part of this decision. For example, suppose you deploy a single storage account to contain all of your tenants' data. If you exceed a specific number of storage operations per second, Azure Storage will reject your application's requests, and all of your tenants will be impacted. This is called *throttling* behavior. It's important that you monitor for throttled requests. For more information, see [Retry guidance for Azure services](#).

Data isolation

When designing a solution that contains multitenant data services, there are usually different options and levels of data isolation, each with their own benefits and tradeoffs. For example:

- When using Azure Cosmos DB, you can deploy separate containers for each tenant, and you can share databases and accounts between multiple tenants. Alternatively, you might consider deploying different databases or even accounts for each tenant, depending on the level of isolation required.
- When using Azure Storage for blob data, you can deploy separate blob containers for each tenant, or you can deploy separate storage accounts.
- When using Azure SQL, you can use separate tables in shared databases, or you can deploy separate databases or servers for each tenant.
- In all Azure services, you can consider deploying resources within a single shared Azure subscription, or you can use multiple Azure subscriptions - perhaps even one per tenant.

There is no single solution that works for every situation. The option you choose depends on a number of factors and the requirements of your tenants. For example, if your tenants need to meet specific compliance or regulatory standards, you might need to apply a higher level of isolation. Similarly, you might have commercial requirements to physically isolate your customers' data, or you might need to enforce isolation to avoid the [Noisy Neighbor problem](#). Additionally, if tenants need to use their own encryption keys, they have individual backup and restore policies, or they need to have

their data stored in different geographical locations, you might need to isolate them from other tenants, or group them with tenants that have similar policies.

Complexity of implementation

It's important to consider the complexity of your implementation. It's good practice to keep your architecture as simple as possible, while still meeting your requirements. Avoid committing to an architecture that will become increasingly complex as you scale, or an architecture that you don't have the resources or expertise to develop and maintain.

Similarly, if your solution doesn't need to scale to a large number of tenants, or if you don't have concerns around performance or data isolation, then it's better to keep your solution simple and avoid adding unnecessary complexity.

A particular concern for multitenant data solutions is the level of customization you support. For example, can a tenant extend your data model or apply custom data rules? Ensure that you design for this requirement upfront. Avoid forking or providing custom infrastructure for individual tenants. Customized infrastructure inhibits your ability to scale, to test your solution, and to deploy updates. Instead, consider using [feature flags](#) and other forms of tenant configuration.

Complexity of management and operations

Consider how you plan to operate your solution, and how your multitenancy approach affects your operations and processes. For example:

- **Management:** Consider cross-tenant management operations, such as regular maintenance activities. If you use multiple accounts, servers, or databases, how will you initiate and monitor the operations for each tenant?
- **Monitoring and metering:** If you monitor or meter your tenants, consider how your solution reports metrics, and whether they can be easily linked to the tenant that triggered the request.
- **Reporting:** Reporting data from across isolated tenants might require that each tenant publishes data to a centralized data warehouse, rather than running queries on each database individually and then aggregating the results.
- **Schema updates:** If you use a database that enforces a schema, plan how you will deploy schema updates across your estate. Consider how your application knows which schema version to use for a specific tenant's database queries.
- **Requirements:** Consider your tenants' high availability requirements (for example, uptime service level agreements, or SLAs) and disaster recovery requirements (for

example, recovery time objectives, or RTOs, and recovery point objectives, or RPOs). If tenants have different expectations, will you be able to meet each tenant's requirements?

- **Migration:** How will you migrate tenants if they need to move to a different type of service, a different deployment, or another region?

Cost

Generally, the higher the density of tenants to your deployment infrastructure, the lower the cost to provision that infrastructure. However, shared infrastructure increases the likelihood of issues like the [Noisy Neighbor problem](#), so consider the tradeoffs carefully.

Approaches and patterns to consider

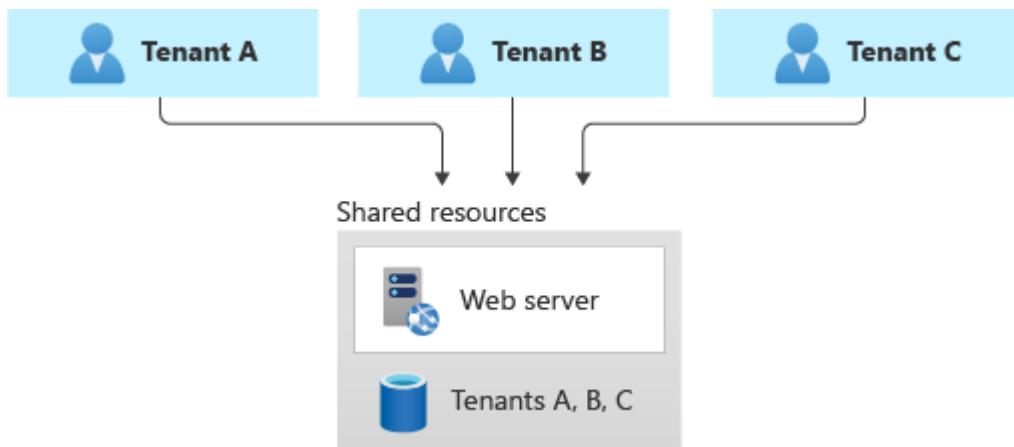
Several design patterns from the Azure Architecture Center are of relevance to multitenant storage and data services. You might choose to follow one pattern consistently. Or, you could consider mixing and matching patterns. For example, you might use a multitenant database for most of your tenants, but deploy single-tenant stamps for tenants who pay more or who have unusual requirements. Similarly, it's often a good practice to scale by using deployment stamps, even when you use a multitenant database or sharded databases within a stamp.

Deployment Stamps pattern

For more information about how the [Deployment Stamps pattern](#) can be used to support a multitenant solution, see [Overview](#).

Shared multitenant databases and file stores

You might consider deploying a shared multitenant database, storage account, or file share, and sharing it across all of your tenants.



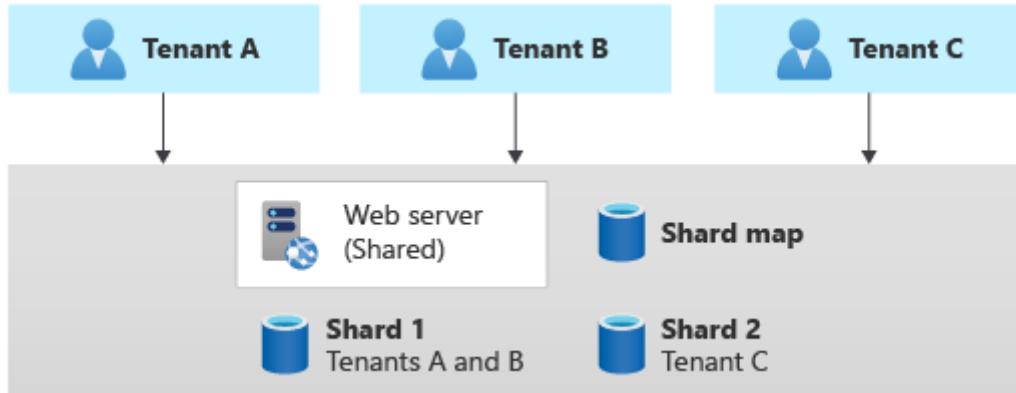
This approach provides the highest density of tenants to infrastructure, so it tends to come at the lowest cost of any approach. It also often reduces the management overhead, because there's a single database or resource to manage, back up, and secure.

However, when you work with shared infrastructure, there are several caveats to consider:

- **Scale limits:** When you rely on a single resource, consider the supported scale and limits of that resource. For example, the maximum size of one database or file store, or the maximum throughput limits, will eventually become a hard blocker, if your architecture relies on a single database. Carefully consider the maximum scale you need to achieve, and compare it to your current and future limits, before you select this pattern.
- **Noisy neighbors:** The [Noisy Neighbor problem](#) might become a factor, especially if you have tenants that are particularly busy or generate higher workloads than others. Considering applying the [Throttling pattern](#) or the [Rate Limiting pattern](#) to mitigate these effects.
- **Monitoring each tenant:** You might have difficulty monitoring the activity and [measuring the consumption](#) for a single tenant. Some services, such as Azure Cosmos DB, provide reporting on resource usage for each request, so this information can be tracked to measure the consumption for each tenant. Other services don't provide the same level of detail. For example, the Azure Files metrics for file capacity are available per file share dimension, only when you use premium shares. However, the standard tier provides the metrics only at the storage account level.
- **Tenant requirements:** Tenants might have different requirements for security, backup, availability, or storage location. If these don't match your single resource's configuration, you might not be able to accommodate them.
- **Schema customization:** When you work with a relational database, or another situation where the schema of the data is important, then tenant-level schema customization is difficult.

Sharding pattern

The [Sharding pattern](#) involves deploying multiple separate databases, called *shards*, that each contain one or more tenants' data. Unlike deployment stamps, shards don't imply that the entire infrastructure is duplicated. You might shard databases without also duplicating or sharding other infrastructure in your solution.



Sharding is closely related to *partitioning*, and the terms are often used interchangeably. Consider the [Horizontal, vertical, and functional data partitioning guidance](#).

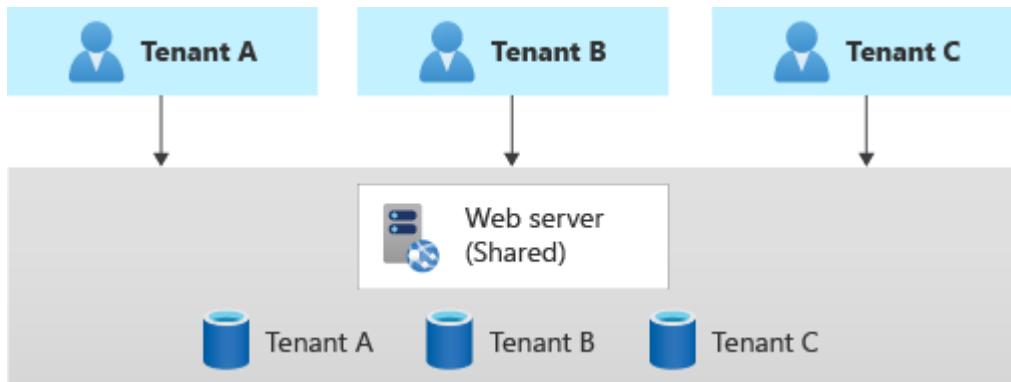
The Sharding pattern can scale to very large numbers of tenants. Additionally, depending on your workload, you might be able to achieve a high density of tenants to shards, so the cost can be attractive. The Sharding pattern can also be used to address [Azure subscription and service quotas, limits and constraints](#).

Some data stores, such as Azure Cosmos DB, provide native support for sharding or partitioning. When you work with other solutions, such as Azure SQL, it can be more complex to build a sharding infrastructure and to route requests to the correct shard, for a given tenant.

Sharding also makes it difficult to support tenant-level configuration differences, and to enable customers to provide their own encryption keys.

Multitenant app with dedicated databases for each tenant

Another common approach is to deploy a single multitenant application, with dedicated databases for each tenant.



In this model, each tenant's data is isolated from the others, and you might be able to support some degree of customization for each tenant.

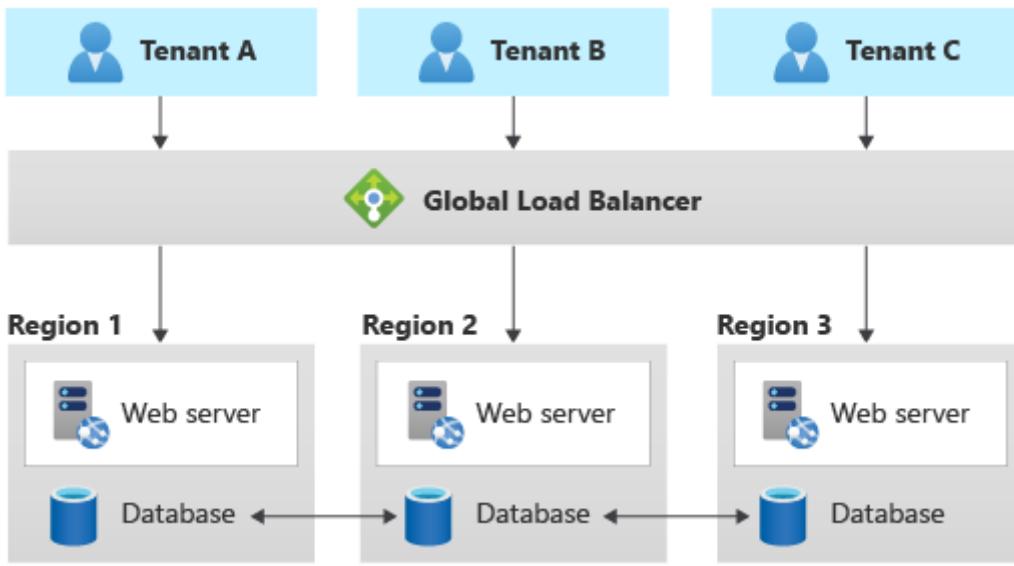
Because you provision dedicated data resources for each tenant, the cost for this approach can be higher than shared hosting models. However, Azure provides several options you can consider, in order to share the cost of hosting individual data resources across multiple tenants. For example, when you work with Azure SQL, you can consider [elastic pools](#). For Azure Cosmos DB, you can [provision throughput for a database](#) and the throughput is shared between the containers in that database, although this approach is not appropriate when you need guaranteed performance for each container.

In this approach, because only the data components are deployed individually for each tenant, you likely can achieve high density for the other components in your solution and reduce the cost of those components.

It's important to use automated deployment approaches when you provision databases for each tenant.

Geodes pattern

The [Geode pattern](#) is designed specifically for geographically distributed solutions, including multitenant solutions. It supports high load and high levels of resiliency. If you implement the Geode pattern, your data tier must be able to replicate the data across geographic regions, and it should support multi-geography writes.



Azure Cosmos DB provides [multi-master writes](#) to support this pattern, and Cassandra supports multi-region clusters. Other data services are generally not able to support this pattern, without significant customization.

Antipatterns to avoid

When you create multitenant data services, it's important to avoid situations that inhibit your ability to scale.

For relational databases, these include:

- **Table-based isolation.** When you work within a single database, avoid creating individual tables for each tenant. A single database won't be able to support very large numbers of tenants when you use this approach, and it becomes increasingly difficult to query, manage, and update data. Instead, consider using a single set of multitenant tables with a tenant identifier column. Alternatively, you can use one of the patterns described above to deploy separate databases for each tenant.
- **Column-level tenant customization.** Avoid applying schema updates that only apply to a single tenant. For example, suppose you have a single multitenant database. Avoid adding a new column to meet a specific tenant's requirements. It might be acceptable for a small number of customizations, but this rapidly becomes unmanageable when you have a large number of customizations to consider. Instead, consider revising your data model to track custom data for each tenant in a dedicated table.
- **Manual schema changes.** Avoid updating your database schema manually, even if you only have a single shared database. It's easy to lose track of the updates you've applied, and if you need to scale out to more databases, it's challenging to identify the correct schema to apply. Instead, build an automated pipeline to

deploy your schema changes, and use it consistently. Track the schema version used for each tenant in a dedicated database or lookup table.

- **Version dependencies.** Avoid having your application take a dependency on a single version of your database schema. As you scale, you might need to apply schema updates at different times for different tenants. Instead, ensure your application version is backwards-compatible with at least one schema version, and avoid destructive schema updates.

Databases

There are some features that can be useful for multitenancy. However, these aren't available in all database services. Consider whether you need these, when you decide on the service to use for your scenario:

- **Row-level security** can provide security isolation for specific tenants' data in a shared multitenant database. This feature is available in Azure SQL and Postgres Flex, but it's not available in other databases, like MySQL or Azure Cosmos DB.
- **Tenant-level encryption** might be required to support tenants that provide their own encryption keys for their data. This feature is available in Azure SQL as part of [Always Encrypted](#). Azure Cosmos DB provides [customer-managed keys at the account level](#) and also [supports Always Encrypted](#).
- **Resource pooling** provides the ability to share resources and cost, between multiple databases or containers. This feature is available in Azure SQL's [elastic pools](#) and [managed instances](#) and in Azure Cosmos DB [database throughput](#), although each option has limitations you should be aware of.
- **Sharding and partitioning** has stronger native support in some services than others. This feature is available in Azure Cosmos DB, by using its [logical and physical partitioning](#), and in [PostgreSQL Hyperscale](#). While Azure SQL doesn't natively support sharding, it provides [sharding tools](#) to support this type of architecture.

Additionally, when you work with relational databases or other schema-based databases, consider where the schema upgrade process should be triggered, when you maintain a fleet of databases. In a small estate of databases, you might consider using a deployment pipeline to deploy schema changes. As you grow, it might be better for your application tier to detect the schema version for a specific database and to initiate the upgrade process.

File and blob storage

Consider the approach you use to isolate data within a storage account. For example, you might deploy separate storage accounts for each tenant, or you might share storage accounts and deploy individual containers. Alternatively, you might create shared blob containers, and then you can use the blob path to separate data for each tenant. Consider [Azure subscription limits and quotas](#), and carefully plan your growth to ensure your Azure resources scale to support your future needs.

If you use shared containers, plan your authentication and authorization strategy carefully, to ensure that tenants can't access each other's data. Consider the [Valet Key pattern](#), when you provide clients with access to Azure Storage resources.

Cost allocation

Consider how you'll [measure consumption and allocate costs to tenants](#), for the use of shared data services. Whenever possible, aim to use built-in metrics instead of calculating your own. However, with shared infrastructure, it becomes hard to split telemetry for individual tenants, and you should consider application-level custom metering.

In general, cloud-native services, like Azure Cosmos DB and Azure Blob Storage, provide more granular metrics to track and model the usage for a specific tenant. For example, Azure Cosmos DB provides the consumed throughput for every request and response.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Paul Burpo](#) | Principal Customer Engineer, FastTrack for Azure
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

For more information about multitenancy and specific Azure services, see:

- [Multitenancy and Azure Storage](#)
- [Multitenancy and Azure SQL Database](#)
- [Multitenancy and Azure Cosmos DB](#)
- [Multitenancy and Azure Database for PostgreSQL](#)

Architectural approaches for messaging in multitenant solutions

Article • 09/16/2022

Asynchronous messaging and event-driven communication are critical assets when building a distributed application that's composed of several internal and external services. When you design a multitenant solution, it's crucial to conduct a preliminary analysis to define how to share or partition messages that pertain to different tenants.

Sharing the same messaging system or event-streaming service can significantly reduce the operational cost and management complexity. However, using a dedicated messaging system for each tenant provides better data isolation, reduces the risk of data leakage, eliminates the [Noisy Neighbor issue](#), and allows to charge back Azure costs to tenants easily.

In this article, you can find a distinction between messages and events, and you'll find guidelines that solution architects can follow when deciding which approach to use for a messaging or eventing infrastructure in a multitenant solution.

Messages, data points, and discrete events

All messaging systems have similar functionalities, transport protocols, and usage scenarios. For example, most of the modern messaging systems support asynchronous communications that use volatile or persistent queues, AMQP and HTTPS transport protocols, at-least-once delivery, and so on. However, by looking more closely at the type of published information and how data is consumed and processed by client applications, we can distinguish between different kinds of messages and events.

There's an essential distinction between services that deliver an event and systems that send a message. For more information, see the following resources:

- [Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus](#)
- [Events, Data Points, and Messages - Choosing the right Azure messaging service for your data ↗](#)

Events

An event is a lightweight notification of a condition or a state change. Events can be discrete units or part of a series. Events are messages that don't generally convey a

publisher's intent other than to inform. An event captures a fact and communicates it to other services or components. A consumer of the event can process the event as it pleases and doesn't fulfill any specific expectations the publisher holds. We can classify events into two main categories:

- **Discrete events** hold information about specific actions that the publishing application has carried out. When using asynchronous event-driven communication, an application publishes a notification event when something happens within its domain. One or more consuming applications needs to be aware of this state change, like a price change in a product catalog application. Consumers subscribe to the events to receive them asynchronously. When a given event happens, the consuming applications might update their domain entities, which can cause more integration events to be published.
- **Series events** carry informational data points, such as temperature readings from devices for analysis or user actions in a click-analytics solution, as elements in an ongoing, continuous stream. An event stream is related to a specific context, like the temperature or humidity reported by a field device. All the events related to the same context have a strict temporal order that matters when processing these events with an event stream processing engine. Analyzing streams of events that carry data points requires accumulating these events in a buffer that spans a desired time window. Or it can be in a selected number of items and then processing these events, by using a near-real-time solution or machine-trained algorithm. The analysis of a series of events can yield signals, like the average of a value measured over a time window that crosses a threshold. Those signals can then be raised as discrete, actionable events.

Discrete events report state changes and are actionable. The event payload has information about what happened, but, in general, it doesn't have the complete data that triggered the event. For example, an event notifies consumers that a reporting application created a new file in a storage account. The event payload might have general information about the file, but it doesn't have the file itself. Discrete events are ideal for serverless solutions that need to scale.

Series events report a condition and are analyzable. Discrete events are time-ordered and interrelated. In some contexts, the consumer needs to receive the complete sequence of events to analyze what happened and to take the necessary action.

Messages

Messages contain raw data that's produced by a service to be consumed or stored elsewhere. Messages often carry information necessary for a receiving service to execute steps in a workflow or a processing chain. An example of this kind of communication is

the [Command pattern](#). The publisher may also expect the receiver(s) of a message to execute a series of actions and to report back the outcome of their processing with an asynchronous message. A contract exists between the message publisher and message receiver(s). For example, the publisher sends a message with some raw data and expects the consumer to create a file from that data and send back a response message when done. In other situations, a sender process could send an asynchronous, one-way message to ask another service to perform a given operation, but with no expectation of getting back an acknowledgment or response message from it.

This kind of contractual message handling is quite different from a publisher who's publishing discrete events to an audience of consumer agents, without any specific expectation of how they will handle these notifications.

Example scenarios

Here is a list of some example multitenant scenarios for messages, data points, and discrete events:

- Discrete events:
 - A music-sharing platform tracks the fact that a specific user in a specific tenant has listened to a particular music track.
 - In an online store web application, the catalog application sends an event using the [Publisher-Subscriber pattern](#) to other applications to notify them that an item price has changed.
 - A manufacturing company pushes real-time information to customers and third parties about equipment maintenance, systems health, and contract updates.
- Data points:
 - A building control system receives telemetry events, such as temperature or humidity readings from sensors that belong to multiple customers.
 - An event-driven monitoring system receives and processes diagnostics logs in a near-real-time fashion from multiple services, such as web servers.
 - A client-side script on a web page collects user actions and sends them to a click-analytics solution.
- Messages:
 - A B2B finance application receives a message to begin processing a tenant's banking records.
 - A long-running workflow receives a message that triggers the execution of the next step.
 - The basket application of an online store application sends a CreateOrder command, by using an asynchronous, persistent message to the ordering application.

Key considerations and requirements

The [deployment and tenancy model](#) that you choose for your solution has a deep impact on security, performance, data isolation, management, and the ability to cross-charge resource costs to tenants. This analysis includes the model that you select for your messaging and eventing infrastructure. In this section, we review some of the key decisions you must make when you plan for a messaging system in your multitenant solution.

Scale

The number of tenants, the complexity of message flows and event streams, the volume of messages, the expected traffic profile, and the isolation level should drive the decisions when planning for a messaging or eventing infrastructure.

The first step consists in conducting exhaustive capacity planning and establishing the necessary maximum capacity for a messaging system in terms of throughput to properly handle the expected volume of messages under regular or peak traffic.

Some service offerings, such as the [Azure Service Bus Premium tier](#), provide resource isolation at the CPU and memory level so that each customer workload runs in isolation. This resource container is called a *messaging unit*. Each premium namespace is allocated at least one messaging unit. You can purchase 1, 2, 4, 8, or 16 messaging units for each Service Bus Premium namespace. A single workload or entity can span multiple messaging units, and you can change the number of messaging units as necessary. The result is a predictable and repeatable performance for your Service Bus-based solution. For more information, see [Service Bus Premium and Standard messaging tiers](#). Likewise, Azure Event Hubs pricing tiers allow you to size the namespace, based on the expected volume of inbound and outbound events. For example, the Premium offering is billed by [processing units](#) (PUs), which correspond to a share of isolated resources (CPU, memory, and storage) in the underlying infrastructure. The effective ingest and stream throughput per PU will depend on the following factors:

- Number of producers and consumers
- Payload size
- Partition count
- Egress request rate
- Usage of Event Hubs Capture, Schema Registry, and other advanced features

For more information, see [Overview of Event Hubs Premium](#).

When your solution handles a considerable number of tenants, and you decide to adopt a separate messaging system for each tenant, you need to have a consistent strategy to automate the deployment, monitoring, alerting, and scaling of each infrastructure, separately from one other.

For example, a messaging system for a given tenant could be deployed during the provisioning process using an infrastructure as code (IaC) tool such as Terraform, Bicep, or Azure Resource Manager (ARM) JSON templates and a DevOps system such as Azure DevOps or GitHub Actions. For more information, see [Architectural approaches for the deployment and configuration of multitenant solutions](#).

The messaging system could be sized with a maximum throughput in messages per unit of time. If the system supports dynamic autoscaling, its capacity could be increased or decreased automatically, based on the traffic conditions and metrics to meet the expected service-level agreement.

Performance predictability and reliability

When designing and building a messaging system for a limited number of tenants, using a single messaging system could be an excellent solution to meet the functional requirements, in terms of throughput, and it could reduce the total cost of ownership. A multitenant application might share the same messaging entities, such as queues and topics across multiple customers. Or they might use a dedicated set of components for each, in order to increase tenant isolation. On the other hand, sharing the same messaging infrastructure across multiple tenants could expose the entire solution to the [Noisy Neighbor issue](#). The activity of one tenant could harm other tenants, in terms of performance and operability.

In this case, the messaging system should be properly sized to sustain the expected traffic load at peak time. Ideally, it should support autoscaling. The messaging system should dynamically scale out when the traffic increases and scale in when the traffic decreases. A dedicated messaging system for each tenant could also mitigate the Noisy Neighbor risk, but managing a large number of messaging systems could increase the complexity of the solution.

A multitenant application can eventually adopt a hybrid approach, where core services use the same set of queues and topics in a single, shared messaging system, in order to implement internal, asynchronous communications. In contrast, other services could adopt a dedicated group of messaging entities or even a dedicated messaging system, for each tenant to mitigate the Noisy Neighbor issue and guarantee data isolation.

When deploying a messaging system to virtual machines, you should co-locate these virtual machines with the compute resources, to reduce the network latency. These virtual machines should not be shared with other services or applications, to avoid the messaging infrastructure to compete for the system resources, such as CPU, memory, and network bandwidth with other systems. Virtual machines can also spread across the [availability zones](#), to increase the intra-region resiliency and reliability of business-critical workloads, including messaging systems. Suppose you decide to host a messaging system, such as [RabbitMQ](#) or [Apache ActiveMQ](#), on virtual machines. In that case, we recommend deploying it to a virtual machine scale set and configuring it for autoscaling, based on metrics such as CPU or memory. This way, the number of agent nodes hosting the messaging system will properly scale out and scale in, based on traffic conditions. For more information, see [Overview of autoscale with Azure virtual machine scale sets](#).

Likewise, when hosting your messaging system on a Kubernetes cluster, consider using node selectors and taints to schedule the execution of its pods on a dedicated node pool, not shared with other workloads, in order to avoid the [Noisy Neighbor issue](#). When deploying a messaging system to a zone-redundant AKS cluster, use [Pod topology spread constraints](#) to control how pods are distributed across your AKS cluster among failure-domains, such as regions, availability zones, and nodes. When hosting a third-party messaging system on AKS, use Kubernetes autoscaling to dynamically scale out the number of worker nodes when the traffic increases. With [Horizontal Pod Autoscaler](#) and [AKS cluster autoscaler](#), node and pod volumes get adjusted dynamically in real-time, to match the traffic condition and to avoid downtimes that are caused by capacity issues. For more information, see [Automatically scale a cluster to meet application demands on Azure Kubernetes Service \(AKS\)](#). Consider using [Kubernetes Event-Driven Autoscaling \(KEDA\)](#), if you want to scale out the number of pods of a Kubernetes-hosted messaging system, such as [RabbitMQ](#) or [Apache ActiveMQ](#), based on the length of a given queue. With KEDA, you can drive the scaling of any container in Kubernetes, based on the number of events needing to be processed. For example, you can use KEDA to scale applications based on the length of an [Azure Service Bus queue](#), of a [RabbitMQ queue](#), or an [ActiveMQ queue](#). For more information on KEDA, see [Kubernetes Event-driven Autoscaling](#).

Isolation

When designing the messaging system for a multitenant solution, you should consider that different types of applications require a different kind of isolation, which is based on the tenants' performance, privacy, and auditing requirements.

- **Multiple tenants can share the same messaging entities, such as queues, topics, and subscriptions, in the same messaging system.** For example, a multitenant application could receive messages that carry data for multiple tenants, from a single [Azure Service Bus](#) queue. This solution can lead to performance and scalability issues. If some of the tenants generate a larger volume of orders than others, this could cause a backlog of messages. This problem, also known as the [Noisy Neighbor issue](#), can degrade the service level agreement in terms of latency for some tenants. The problem is more evident if the consumer application reads and processes messages from the queue, one at a time, in a strict chronological order, and if the time necessary to process a message depends on the number of items in the payload. When sharing one or more queue resources across multiple tenants, the messaging infrastructure and processing systems should be able to accommodate the scale and throughput requirements-based expected traffic. This architectural approach is well-suited in those scenarios where a multitenant solution adopts a single pool of worker processes, in order to receive, process, and send messages for all the tenants.
- **Multiple tenants share the same messaging system but use different topic or queue resources.** This approach provides better isolation and data protection at a higher cost, increased operational complexity, and lower agility because system administrators have to provision, monitor, and maintain a higher number of queue resources. This solution ensures a consistent and predictable experience for all tenants, in terms of performance and service-level agreements, as it prevents any tenant from creating a bottleneck that can impact the other tenants.
- **A separate, dedicated messaging system is used for each tenant.** For example, a multitenant solution can use a distinct [Azure Service Bus](#) namespace for each tenant. This solution provides the maximum degree of isolation, at a higher complexity and operational cost. This approach guarantees consistent performance and allows for fine-tuning the messaging system, based on specific tenant needs. When a new tenant is onboarded, in addition to a fully dedicated messaging system, the provisioning application might also need to create a separate managed identity or a service principal with the proper RBAC permissions to access the newly created namespace. For example, when a Kubernetes cluster is shared by multiple instances of the same SaaS application, one for each tenant, and each running in a separate namespace, each application instance may use a different service principal or managed identity to access a dedicated messaging system. In this context, the messaging system could be a fully-managed PaaS service, such as an [Azure Service Bus](#) namespace, or a Kubernetes-hosted messaging system, such as [RabbitMQ](#). When deleting a tenant from the system, the application should remove the messaging system and any dedicated resource that was priorly created for the tenant. The main disadvantage of this approach is

that it increases operational complexity and reduces agility, especially when the number of tenants grows over time.

Review the following additional considerations and observations:

- If tenants need to use their own key to encrypt and decrypt messages, a multitenant solution should provide the option to adopt a separate Azure Service Bus Premium namespace for each tenant. For more information, see [Configure customer-managed keys for encrypting Azure Service Bus data at rest](#).
- If tenants need a high level of resiliency and business continuity, a multitenant solution should provide the ability to provision a Service Bus Premium namespace with geo-disaster recovery and [availability zones](#) enabled. For more information, see [Azure Service Bus Geo-disaster recovery](#).
- When using separate queue resources or a dedicated messaging system for each tenant, it's reasonable to adopt a separate pool of worker processes, for each of them to increase the data isolation level and reduce the complexity of dealing with multiple messaging entities. Each instance of the processing system could adopt different credentials, such as a connection string, a service principal, or a managed identity, in order to access the dedicated messaging system. This approach provides a better security level and isolation between tenants, but it requires an increased complexity in identity management.

Likewise, an event-driven application can provide different levels of isolation:

- An event-driven application receives events from multiple tenants, via a single, shared [Azure Event Hubs](#) instance. This solution provides a high level of agility, because onboarding a new tenant does not require creating a dedicated event-ingestion resource, but it provides a low data protection level, as it inter-mingles messages from multiple tenants in the same data structure.
- Tenants can opt for a dedicated event hub or Kafka topic to avoid the [Noisy Neighbor issue](#) and to meet their data isolation requirements, when events must not be co-mingled with those of other tenants, for security and data protection.
- If tenants need a high level of resiliency and business continuity, a multitenant should provide the ability to provision an Event Hubs Premium namespace, with geo-disaster recovery and [availability zones](#) enabled. For more information, see [Azure Event Hubs - Geo-disaster recovery](#).
- Dedicated Event Hubs, with Event Hubs Capture enabled, should be created for those customers that need to archive events to an Azure Storage Account, for regulatory compliance reasons and obligations. For more information, see [Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage](#).

- A single multitenant application can send notifications to multiple internal and external systems, by using [Azure Event Grid](#). In this case, a separate Event Grid subscription should be created for each consumer application. If your application makes use of multiple Event Grid instances to send notifications to a large number of external organizations, consider using *event domains*, which allow an application to publish events to thousands of topics, such as one for each tenant. For more information, see [Understand event domains for managing Event Grid topics](#).

Complexity of implementation

When designing a multitenant solution, it's essential to consider how the system will evolve in the medium to long term, in order to prevent its complexity from growing over time, until it is necessary to redesign part of or the entire solution. This redesign will help you cope with an increasing number of tenants. When designing a messaging system, you should consider the expected growth in message volumes and tenants, in the next few years, and then create a system that can scale out, in order to keep up with the predicted traffic and to reduce the complexity of operations, such as provisioning, monitoring, and maintenance.

The solution should automatically create or delete the necessary messaging entities any time that a tenant application is provisioned or unprovisioned, without the need for manual operations.

A particular concern for multitenant data solutions is the level of customization that you support. Any customization should be automatically configured and applied by the application provisioning system (such as a DevOps system), which is based on a set of initial parameters, whenever a single-tenant or multitenant application is deployed.

Complexity of management and operations

Plan from the beginning how you intend to operate, monitor, and maintain your messaging and eventing infrastructure and how your multitenancy approach affects your operations and processes. For example, consider the following possibilities:

- You might plan to host the messaging system that's used by your application in a dedicated set of virtual machines, one for each tenant. If so, how do you plan to deploy, upgrade, monitor, and scale out these systems?
- Likewise, if you containerize and host your messaging system in a shared Kubernetes cluster, how do you plan to deploy, upgrade, monitor, and scale out individual messaging systems?

- Suppose you share a messaging system across multiple tenants. How can your solution collect and report the per-tenant usage metrics or throttle the number of messages that each tenant can send or receive?
- When your messaging system leverages a PaaS service, such as Azure Service Bus, you should ask the following question:
 - How can you customize the pricing tier for each tenant, based on the features and isolation level (shared or dedicated) that are requested by the tenant?
 - How can you create a per-tenant manage identity and Azure role assignment to assign the proper permissions only to the messaging entities, such as queues, topics, and subscriptions, that the tenant can access? For more information, see [Authenticate a managed identity with Azure Active Directory to access Azure Service Bus resources](#).
- How will you migrate tenants, if they need to move to a different type of messaging service, a different deployment, or another region?

Cost

Generally, the higher the density of tenants to your deployment infrastructure, the lower the cost to provision that infrastructure. However, shared infrastructure increases the likelihood of issues like the [Noisy Neighbor issue](#), so consider the tradeoffs carefully.

Approaches and patterns to consider

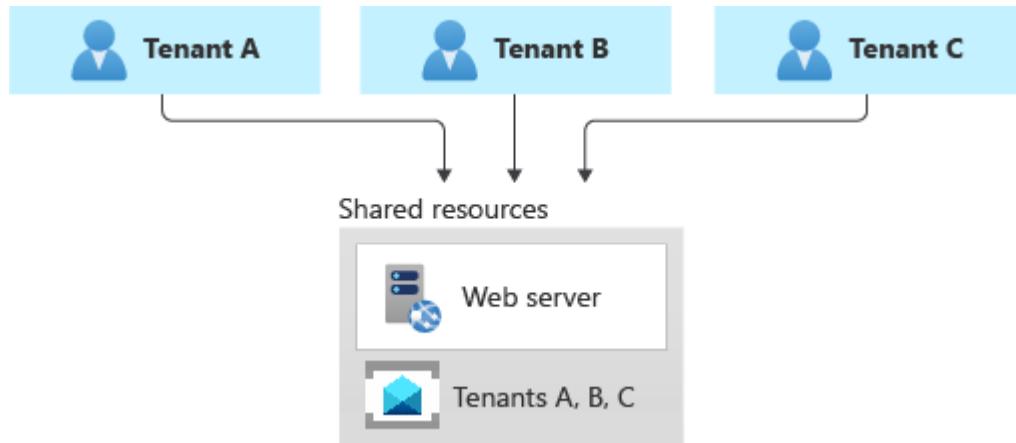
Several [Cloud Design Patterns](#) from the Azure Architecture Center can be applied to a multitenant messaging system. You might choose to follow one or more patterns consistently, or you could consider mixing and matching patterns, based on your needs. For example, you might use a multitenant Service Bus namespace for most of your tenants, but then you might deploy a dedicated, single-tenant Service Bus namespace for those tenants who pay more or who have higher requirements, in terms of isolation and performance. Similarly, it's often a good practice to scale by using deployment stamps, even when you use a multitenant Service Bus namespace or dedicated namespaces within a stamp.

Deployment Stamps pattern

For more information about the Deployment Stamps pattern and multitenancy, see [the Deployment Stamps pattern section of Architectural approaches for multitenancy](#). For more information about tenancy models, see [Tenancy models to consider for a multitenant solution](#).

Shared messaging system

You might consider deploying a shared messaging system, such as Azure Service Bus, and sharing it across all of your tenants.



This approach provides the highest density of tenants to the infrastructure, so it reduces the overall total cost of ownership. It also often reduces the management overhead, since there's a single messaging system or resource to manage and secure.

However, when you share a resource or an entire infrastructure across multiple tenants, consider the following caveats:

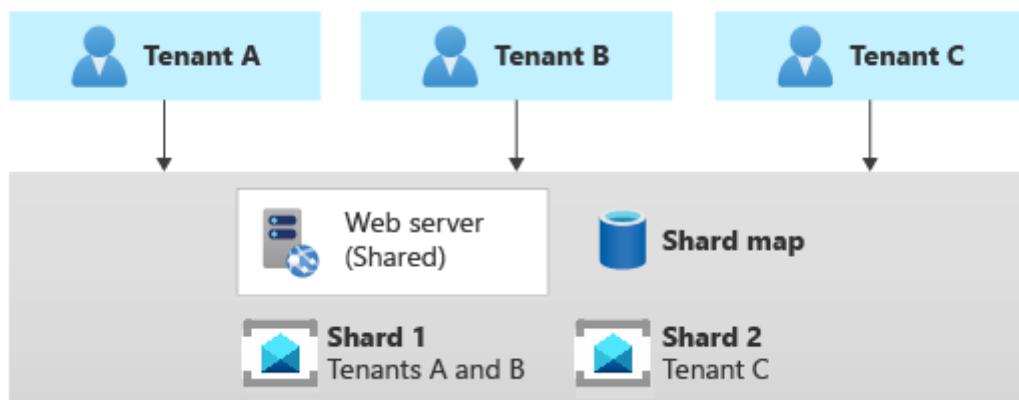
- Always keep in mind and consider the constraints, scaling capabilities, quotas, and limits of the resource in question. For example, the maximum number of [Service Bus namespaces in an Azure subscription](#), the maximum number of [Event Hubs in a single namespace](#), or the maximum throughput limits, might eventually become a hard blocker, if and when your architecture grows to support more tenants. Carefully consider the maximum scale that you need to achieve in terms of the number of namespaces per single Azure subscription, or queues per single namespace. Then compare your current and future estimates to the existing quotas and limits of the messaging system of choice, before you select this pattern.
- As mentioned in the above sections, the [Noisy Neighbor problem](#) might become a factor, when you share a resource across multiple tenants, especially if some are particularly busy or if they generate higher traffic than others. In this case, consider applying the [Throttling pattern](#) or the [Rate Limiting pattern](#) to mitigate these effects. For example, you could limit the maximum number of messages that a single tenant can send or receive in the unit of time.
- You might have difficulty monitoring the activity and [measuring the resource consumption](#) for a single tenant. Some services, such as Azure Service Bus, charge for messaging operations. Hence, when you share a namespace across multiple tenants, your application should be able to keep track of the number of messaging

operations done on behalf of each tenant and the chargeback costs to them. Other services don't provide the same level of detail.

Tenants may have different requirements for security, intra-region resiliency, disaster recovery, or location. If these don't match your messaging system configuration, you might not be able to accommodate them just with a single resource.

Sharding pattern

The [Sharding pattern](#) involves deploying multiple messaging systems, called *shards*, which contain one or more tenants' messaging entities, such as queues and topics. Unlike deployment stamps, shards don't imply that the entire infrastructure is duplicated. You might shard messaging systems without also duplicating or sharding other infrastructure in your solution.



Every messaging system or *shard* can have different characteristics in terms of reliability, SKU, and location. For example, you could shard your tenants across multiple messaging systems with different characteristics, based on their location or needs in terms of performance, reliability, data protection, or business continuity.

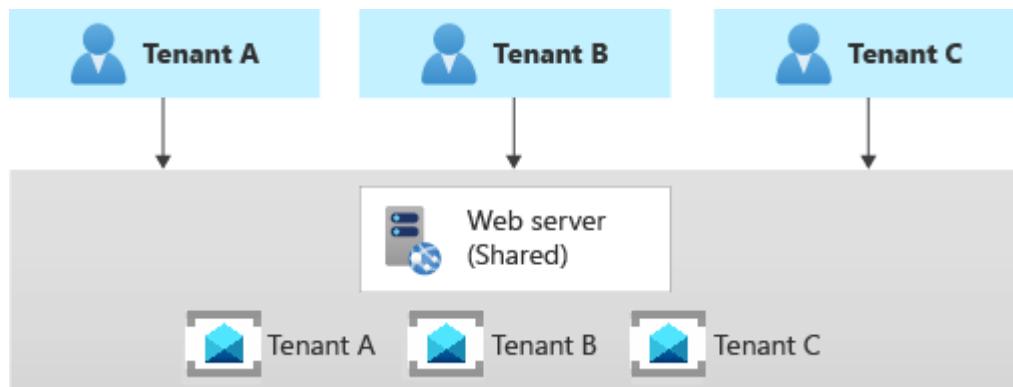
When using the sharding pattern, you need to use a [sharding strategy](#), in order to map a given tenant to the messaging system that contains its queues. The [lookup strategy](#) uses a map to individuate the target messaging system, based on the tenant name or ID. Multiple tenants might share the same shard, but the messaging entities used by a single tenant won't be spread across multiple shards. The map can be implemented with a single dictionary that maps the tenant's name to the name or reference of the target messaging system. The map can be stored in a distributed cache accessible, by all the instances of a multitenant application, or in a persistent store, such as a table in a relational database or a table in a storage account.

The Sharding pattern can scale to very large numbers of tenants. Additionally, depending on your workload, you might be able to achieve a high density of tenants to

shards, so the cost can be attractive. The Sharding pattern can also be used to address Azure subscription and service quotas, limits, and constraints.

Multitenant app with dedicated messaging system for each tenant

Another common approach is to deploy a single multitenant application, with dedicated messaging systems for each tenant. In this tenancy model, you have some shared components, such as computing resources, while other services are provisioned, and managed using a single-tenant, dedicated deployment approach. For example, you could build a single application tier, and then deploy individual messaging systems for each tenant, as shown in the following illustration.



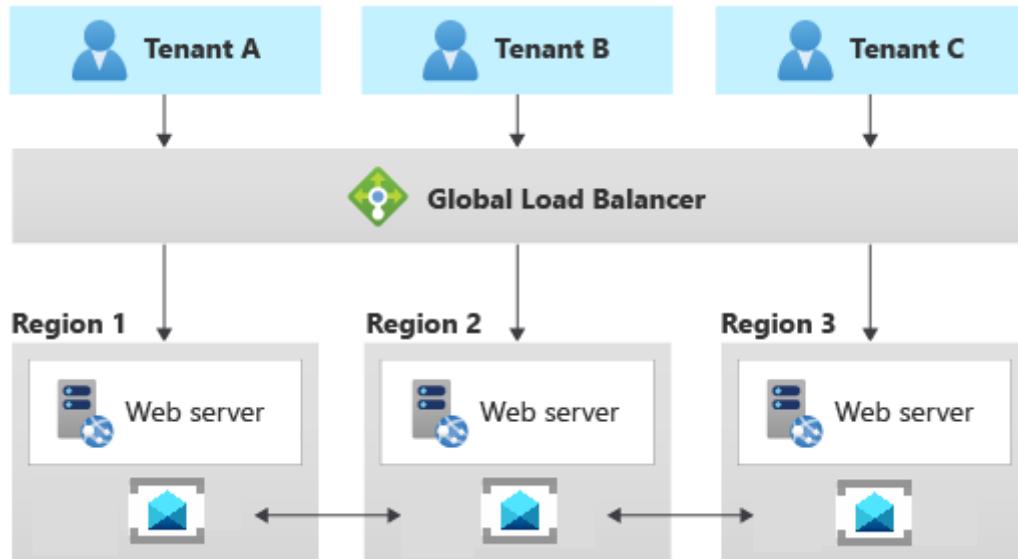
Horizontally partitioned deployments can help you mitigate a noisy-neighbor problem, if you've identified that most of the load on your system is due to specific components that you can deploy separately for each tenant. For example, you may need to use a separate messaging or event streaming system for each tenant because a single instance is not enough to keep up with traffic that's generated by multiple tenants. When using a dedicated messaging system for each tenant, if a single tenant causes a large volume of messages or events, this might affect the shared components but not other tenants' messaging systems.

Because you provision dedicated resources for each tenant, the cost for this approach can be higher than a shared hosting model. On the other hand, it's easier to charge back resource costs of a dedicated system to the unique tenant that makes use of it, when adopting this tenancy model. This approach allows you to achieve high density for other services, such as computing resources, and it reduces these components' costs.

With a horizontally partitioned deployment, you need to adopt an automated process for deploying and managing a multitenant application's services, especially those used by a single tenant.

Geodes pattern

The Geode pattern involves deploying a collection of backend services into a set of geographical nodes. Each can service any request for any client in any region. This pattern allows you to serve requests in an active-active style, which improves latency and increases availability, by distributing request processing around the globe.



Azure Service Bus and Azure Event Hubs support metadata disaster recovery, across primary and secondary disaster recovery namespaces and across separate regions and availability zones, in order to provide support for the best intra-region resiliency. Also, Azure Service Bus and Azure Event Hubs implement a set of federation patterns that actively replicate the same logical topic, queue, or event hub to be available in multiple namespaces, eventually located in different regions. For more information, see the following resources:

- [Message replication and cross-region federation](#)
- [Message replication tasks patterns](#)
- [Multi-site and multi-region federation](#)
- [Event replication tasks patterns](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

- [Clemens Vasters](#) | Principal Architect, Messaging Services and Standards
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

Next steps

For more information about messaging design patterns, see the following resources:

- [Claim-Check pattern](#)
- [Competing Consumers pattern](#)
- [Event Sourcing pattern](#)
- [Pipes and Filters pattern](#)
- [Publisher-Subscriber pattern](#)
- [Sequential Convoy pattern](#)

Architectural approaches for identity in multitenant solutions

Article • 10/10/2023

Almost all multitenant solutions require an identity system. In this article, we discuss common components of identity, including both authentication and authorization, and we discuss how these components can be applied in a multitenant solution.

ⓘ Note

Review [Architectural considerations for identity in a multitenant solution](#) to learn more about the key requirements and decisions that you need to make, before you start to build an identity system for a multitenant solution.

Authentication

Authentication is the process by which a user's identity is established. When you build a multitenant solution, there are special considerations and approaches for several aspects of the authentication process.

Federation

You might need to federate with other identity providers (IdPs). Federation can be used to enable the following scenarios:

- Social login, such as by enabling users to use their Google, Facebook, GitHub, or personal Microsoft account.
- Tenant-specific directories, such as by enabling tenants to federate your application with their own identity providers, so they don't need to manage accounts in multiple places.

For general information about federation, see the [Federated Identity pattern](#).

If you choose to support tenant-specific identity providers, ensure you clarify which services and protocols you need to support. For example, will you support the OpenID Connect protocol and the Security Assertion Markup Language (SAML) protocol? Or, will you only support federating with Microsoft Entra instances?

When you implement any identity provider, consider any scale and limits that might apply. For example, if you use Azure Active Directory (Azure AD) B2C as your own identity provider, you might need to deploy custom policies to federate with certain types of tenant identity providers. Azure AD B2C [limits the number of custom policies](#) that you can deploy, which might limit the number of tenant-specific identity providers that you can federate with.

You can also consider providing federation as a feature that only applies to customers at a higher [product tier](#).

Single sign-on

Single sign-on experiences enable users to switch between applications seamlessly, without being prompted to reauthenticate at each point.

When users visit an application, the application directs them to an IdP. If the IdP sees they have an existing session, it issues a new token without requiring the users to interact with the login process. A federated identity model support single sign-on experiences, by enabling users to use a single identity across multiple applications.

In a multitenant solution, you might also enable another form of single sign-on. If users are authorized to work with data for multiple tenants, you might need to provide a seamless experience when the users change their context from one tenant to another. Consider whether you need to support seamless transitions between tenants, and if so, whether your identity provider needs to reissue tokens with specific tenant claims. For example, a user who signed into the Azure portal can switch between different Microsoft Entra directories, which causes reauthentication, and it reissues the token from the newly selected Microsoft Entra instance.

Sign-in risk evaluation

Modern identity platforms support a risk evaluation during the sign-in process. For example, if a user signs in from an unusual location or device, the authentication system might require extra identity checks, such as multifactor authentication (MFA), before it allows the sign-in request to proceed.

Consider whether your tenants might have different risk policies that need to be applied during the authentication process. For example, if you have some tenants in a highly regulated industry, they might have different risk profiles and requirements to tenants who work in less regulated environments. Or, you might choose to allow tenants at higher pricing tiers to specify more restrictive sign-in policies than tenants who purchase a lower tier of your service.

If you need to support different risk policies for each tenant, your authentication system needs to know which tenant the user is signing into, so that it can apply the correct policies.

If your IdP includes these capabilities, consider using the IdP's native sign-in risk evaluation features. These features can be complex and error-prone to implement yourself.

Alternatively, if you federate to tenants' own identity providers, then their own risky sign-in mitigation policies can be applied, and they can control the policies and controls that should be enforced. However, it's important to avoid inadvertently adding unnecessary burden to the user, such as by requiring two MFA challenges - one from the user's home identity provider and one from your own. Ensure you understand how federation interacts with each of your tenants' identity providers and the policies they've applied.

Impersonation

Impersonation enables a user to assume the identity of another user, without using that user's credentials.

In general, impersonation is dangerous, and it can be difficult to implement and control. However, in some scenarios, impersonation is a requirement. For example, if you operate software as a service (SaaS), your helpdesk personnel might need to assume a user's identity, so that they can sign in as the user and troubleshoot an issue.

If you choose to implement impersonation, consider how you audit its use. Ensure that your logs include both the actual user who performed the action and the identifier of the user they impersonated.

Some identity platforms support impersonation, either as a built-in feature or by using custom code. For example, [in Azure AD B2C, you can add a custom claim](#) for the impersonated user ID, or you can replace the subject identifier claim in the tokens that are issued.

Authorization

Authorization is the process of determining what a user is allowed to do.

Authorization data can be stored in several places, including in the following locations:

- **In your identity provider.** For example, if you use Microsoft Entra ID as your identity provider, you can use features like [app roles](#) and [groups](#) to store

authorization information. Your application can then use the associated token claims to enforce your authorization rules.

- **In your application.** You can build your own authorization logic, and then store information about what each user can do in a database or similar storage system. You can then design fine-grained controls for role-based or resource-level authorization.

In most multitenant solutions, role and permission assignments are managed by the tenant or customer, not by you as the vendor of the multitenant system.

For more information, see [Application roles](#).

Add tenant identity and role information to tokens

Consider which part, or parts, of your solution should perform authorization requests, including determining whether a user is allowed to work with data from a specific tenant.

A common approach is for your identity system to embed a tenant identifier claim into a token. This approach enables your application to inspect the claim and verify that the users are working with the tenant that they're allowed to access. If you use the role-based security model, then you might choose to extend the token with information about the role a user has within the tenant.

However, if a single user is allowed to access multiple tenants, you might need a way for your users to signal which tenant they plan to work with during the login process. After they select their active tenant, the IdP can include the correct tenant identifier claim and role for that tenant, within the token it issues. You also need to consider how users can switch between tenants, which requires issuing a new token.

Application-based authorization

An alternative approach is to make the identity system agnostic to tenant identifiers and roles. The users are identified using their credentials or a federation relationship, and tokens don't include a tenant identifier claim. A separate list or database contains which users have been granted access to each tenant. Then, the application tier can verify whether the specified user should be allowed to access the data for a specific tenant, based on looking up that list.

Use Microsoft Entra ID or Azure AD B2C

Microsoft provides Microsoft Entra ID and Azure AD B2C, which are managed identity platforms that you can use within your own multitenant solution.

Many multitenant solutions are software as a service (SaaS). Your choice of whether to use Microsoft Entra ID or Azure AD B2C depends, in part, on how you define your tenants or customer base.

- If your tenants or customers are organizations, they might already use Microsoft Entra ID for services like Office 365, Microsoft Teams, or for their own Azure environments. You can create a [multitenant application](#) in your own Microsoft Entra directory, to make your solution available to other Microsoft Entra directories. You can even list your solution in the [Azure Marketplace](#) and make it easily accessible to organizations who use Microsoft Entra ID.
- If your tenants or customers don't use Microsoft Entra ID, or if they're individuals rather than organizations, then consider using Azure AD B2C. Azure AD B2C provides a set of features to control how users sign up and sign in. For example, you can restrict access to your solution just to users that you've already invited, or you might allow for self-service sign-up. Use [custom policies](#) in Azure AD B2C to fully control how users interact with the identity platform. You can use [custom branding](#), and you can [federate Azure AD B2C with your own Microsoft Entra tenant](#), to enable your own staff to sign in. Azure AD B2C also enables [federation with other identity providers](#).
- Some multitenant solutions are intended for both situations listed above. Some tenants might have their own Microsoft Entra tenants, and others might not. You can also use Azure AD B2C for this scenario, and use [custom policies to allow user sign-in from a tenant's Microsoft Entra directory](#). However, if you use custom policies to establish federation between tenants, ensure that you [consider the limits on the number of custom policies](#) that a single Azure AD B2C directory can use.

For more information, see [Considerations for using Azure Active Directory B2C in a multitenant architecture](#).

Antipatterns to avoid

Building or running your own identity system

Building a modern identity platform is complex. There are a range of protocols and standards to support, and it's easy to incorrectly implement a protocol and expose a security vulnerability. Standards and protocols change, and you also need to continually update your identity system to mitigate attacks and to support recent security features.

It's also important to ensure that an identity system is resilient, because any downtime can have severe consequences for the rest of your solution. Additionally, in most situations, implementing an identity provider doesn't add a benefit to the business, and it's simply a necessary part of implementing a multitenant service. It's better to instead use a specialized identity system that's built, operated, and secured by experts.

When you run your own identity system, you need to store password hashes or other forms of credentials, which become a tempting target for attackers. Even hashing and salting passwords is often insufficient protection, because the computational power that's available to attackers can make it possible to compromise these forms of credentials.

When you run an identity system, you're also responsible for generating and distributing MFA or one-time password (OTP) codes. These requirements then mean you need a mechanism to distribute these codes, by using SMS or email. Furthermore, you're responsible for detecting both targeted and brute-force attacks, throttling sign-in attempts, auditing, and so on.

Instead of building or running your own identity system, it's a good practice to use an off-the-shelf service or component. For example, consider using Microsoft Entra ID or Azure AD B2C, which are managed identity platforms. Managed identity platform vendors take responsibility to operate the infrastructure for their platforms, and typically to support the current identity and authentication standards.

Failing to consider your tenants' requirements

Tenants often have strong opinions about how identity should be managed for the solutions they use. For example, many enterprise customers require federation with their own identity providers, to enable single sign-on experiences and to avoid managing multiple sets of credentials. Other tenants might require multifactor authentication, or other forms of protection around the sign-in processes. If you haven't designed for these requirements, it can be challenging to retrofit them later.

Ensure you understand your tenants' identity requirements, before you finalize the design of your identity system. Review [Architectural considerations for identity in a multitenant solution](#), to understand some specific requirements that often emerge.

Conflating users and tenants

It's important to clearly consider how your solution defines a user and a tenant. In many situations, the relationship can be complex. For example, a tenant might contain multiple users, and a single user might join multiple tenants.

Ensure you have a clear process for tracking the tenant context, within your application and requests. In some situations, this process might require you to include a tenant identifier in every access token, and for you to validate the tenant identifier on each request. In other situations, you store the tenant authorization information separately from the user identities, and you use a more complex authorization system, to manage which users can perform which operations against which tenants.

Tracking the tenant context of a user or token is applicable to any [tenancy model](#), because a user identity always has a tenant context within a multitenant solution. It's even a good practice to track tenant context when you deploy independent stamps for a single tenant, which future-proofs your codebase for other forms of multitenancy.

Conflating role and resource authorization

It's important that you select an appropriate authorization model for your solution. Role-based security approaches can be simple to implement, but resource-based authorization provides more fine-grained control. Consider your tenants' requirements, and whether your tenants need to authorize some users to access specific parts of your solution, and not other parts.

Failing to write audit logs

Audit logs are an important tool for understanding your environment and how users are implementing your system. By auditing every identity-related event, you can often determine whether your identity system is under attack, and you can review how your system is being used. Ensure you write and store audit logs within your identity system. Consider whether your solution's identity audit logs should be made available to tenants to review.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirsingh](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Jelle Druyts](#) | Principal Customer Engineer, FastTrack for Azure
- [Sander van den Hoven](#) | Senior Partner Technology Strategist
- [Nick Ward](#) | Senior Cloud Solution Architect

Next steps

Review [Architectural considerations for identity in a multitenant solution](#).

Architectural approaches for tenant integration and data access

Article • 05/18/2023

It's common for systems to integrate together, even across organizational boundaries. When you build a multitenant solution, you might have requirements to send data back to your tenants' systems, or to receive data from those systems. In this article, we outline the key considerations and approaches for architecting and developing integrations for a multitenant solution.

ⓘ Note

If you provide multiple integration points, it's best to consider each one independently. Often, different integration points have different requirements and are designed differently, even if they're connecting the same systems together in multiple different ways.

Key considerations and requirements

Direction of data flow

It's important to understand the direction in which your data flows. The data flow direction affects several aspects of your architecture, such as how you manage identity and your solution's networking topology. There are two common data flows:

- **Export**, which means the data flows from your multitenant system to your individual tenants' systems.
- **Import**, which means data comes from your tenants' systems into your multitenant system.

It's also important to consider the networking data flow direction, which doesn't necessarily correspond to the logical data flow direction. For example, you might initiate an outbound connection to a tenant so that you can import the data from the tenant's system.

Full or user-delegated access

In many systems, access to certain data is restricted to specific users. The data that one user can access might not be the same as the data that another user can access. It's important to consider whether you expect to work with complete data sets, or if the data sets you import or export are based on what a specific user has permission to access.

For example, consider Microsoft Power BI, which is a multitenant service that provides reporting and business intelligence on top of customer-owned data stores. When you configure Power BI, you configure *data sources* to pull data from databases, APIs, and other data stores. You can configure data stores in two different ways:

- **Import all the data from the underlying data store.** This approach requires that Power BI is provided with credentials for an identity that can access the complete data store. Then, Power BI administrators can separately configure permissions to the imported data after it's imported into Power BI. Power BI enforces the permissions.
- **Import a subset of data from the underlying data store, based on a user's permissions.** When a user creates the data source, they use their credentials and the associated permissions. The exact subset of data that Power BI imports depends on the access level of the user who created the data source.

Both approaches have valid use cases, so it's important to clearly understand your tenants' requirements.

If you work with full data sets, the source system effectively treats the destination system as a *trusted subsystem*. For this type of integration, you should also consider using a *workload identity* instead of a user identity. A workload identity is a system identity that doesn't correspond to a single user. The workload identity is granted a high level of permission to the data in the source system.

Alternatively, if you work with user-scoped data, then you might need to use an approach like *delegation* to access the correct subset of data from the data set. Then, the destination system effectively gets the same permission as a specific user. For more information on user delegation, see the [Delegated user access](#) approach below. If you use delegation, consider how you'll handle scenarios where a user is deprovisioned or their permissions change.

Real-time or batch

Consider whether you'll be working with real-time data, or if the data will be sent in batches.

For real-time integrations, these approaches are common:

- **Request/response** is where a client initiates a request to a server and receives a response. Typically, request/response integrations are implemented by using APIs or webhooks. Requests might be *synchronous*, where they wait for acknowledgment and a response. Alternatively, requests can be *asynchronous*, using something like the [Asynchronous request-reply pattern](#) to wait for a response.
- **Loosely coupled communication** is often enabled through messaging components that are designed for loosely coupling systems together. For example, Azure Service Bus provides message queuing capabilities, and Azure Event Grid and Event Hubs provide eventing capabilities. These components are often used as part of integration architectures.

In contrast, batch integrations are often managed through a background job, which might be triggered at certain times of the day. Commonly, batch integrations take place through a *staging location*, such as a blob storage container, because the data sets exchanged can be large.

Data volume

It's important to understand the volume of data that you exchange through an integration, because this information helps you to plan for your overall system capacity. When you plan your system's capacity, remember that different tenants might have different volumes of data to exchange.

For real-time integrations, you might measure volume as the number of transactions over a specified period of time. For batch integrations, you might measure volume either as the number of records exchanged or the amount of data in bytes.

Data formats

When data is exchanged between two parties, it's important they both have a clear understanding of how the data will be formatted and structured. Consider the following parts of the data format:

- The file format, such as JSON, Parquet, CSV, or XML.
- The schema, such as the list of fields that will be included, date formats, and nullability of fields.

When you work with a multitenant system, if possible, it's best to standardize and use the same data format for all of your tenants. That way, you avoid having to customize and retest your integration components for each tenant's requirements. However, in some situations, you might need to use different data formats for communicating with

different tenants, and so you might need to implement multiple integrations. See the section, [Composable integration components](#), for an approach that can help to simplify this kind of situation.

Access to tenants' systems

Some integrations require you to make a connection to your tenant's systems or data stores. When you connect to your tenant's systems, you need to carefully consider both the networking and identity components of the connection.

Network access

Consider the network topology for accessing your tenant's system, which might include the following options:

- **Connect across the internet.** If you connect across the internet, how will the connection be secured, and how will the data be encrypted? If your tenants plan to restrict based on your IP addresses, ensure that the Azure services that your solution uses can support static IP addresses for outbound connections. For example, consider using [NAT Gateway](#) to provide static IP addresses, if necessary.
- **Agents**, which are deployed into a tenant's environment, can provide a flexible approach and can help you avoid the need for your tenants to allow inbound connections.
- **Relays**, such as [Azure Relay](#), also provide an approach to avoid inbound connections.

For more information, see the guidance on [networking approaches for multitenancy](#).

Authentication

Consider how you authenticate with each tenant when you initiate a connection.

Consider the following approaches:

- **Secrets**, such as API keys or certificates. It's important to plan how you'll securely manage your tenants' credentials. Leakage of your tenants' secrets could result in a major security incident, potentially impacting many tenants.
- **Azure Active Directory (Azure AD) tokens**, where you use a token issued by the tenant's own Azure AD instance. The token might be issued directly to your workload by using a multitenant Azure AD application registration or a specific service principal. Alternatively, your workload can request delegated permission to access resources on behalf of a specific user within the tenant's directory.

Whichever approach you select, ensure that your tenants follow the principle of least privilege and avoid granting your system unnecessary permissions. For example, if your system only needs to read data from a tenant's data store, then the identity that your system uses shouldn't have write permissions.

Tenants' access to your systems

If tenants need to connect to your system, consider providing dedicated APIs or other integration points, which you can then model as part of the surface area of your solution.

In some situations, you might decide to provide your tenants with direct access to your Azure resources. Consider the ramifications carefully and ensure you understand how to grant access to tenants in a safe manner. For example, you might use one of the following approaches:

- Use the [Valet Key pattern](#), which involves using security measures like shared access signatures to grant restricted access to certain Azure resources.
- Use dedicated resources for integration points, such as a dedicated storage account. It's a good practice to keep integration resources separated from your core system resources. This approach helps you to minimize the *blast radius* of a security incident. It also ensures that, if a tenant accidentally initiates large numbers of connections to the resource and exhausts its capacity, then the rest of your system continues to run.

Compliance

When you start to interact directly with your tenants' data, or transmit that data, it's critical that you have a clear understanding of your tenants' [governance and compliance requirements](#).

Approaches and patterns to consider

Expose APIs

Real-time integrations commonly involve exposing APIs to your tenants or other parties to use. APIs require special considerations, especially when used by external parties.

Consider the following questions:

- Who is granted access to the API?
- How will you authenticate the API's users?

- Is there a limit to the number of requests that an API user can make over a period of time?
- How will you provide information about your APIs and documentation for each API? For example, do you need to implement a developer portal?

A good practice is to use an API gateway, such as [Azure API Management](#), to handle these concerns and many others. API gateways give you a single place to implement policies that your APIs follow, and they simplify the implementation of your backend API systems.

Valet Key pattern

Occasionally, a tenant might need direct access to a data source, such as Azure Storage. Consider following the [Valet Key pattern](#) to share data securely and to restrict access to the data store.

For example, you could use this approach when batch exporting a large data file. After you've generated the export file, you can save it to a blob container in Azure Storage, and then generate a time-bound, read-only shared access signature. This signature can be provided to the tenant, along with the blob URL. The tenant can then download the file from Azure Storage until the signature's expiry.

Similarly, you can generate a shared access signature with permissions to write to a specific blob. When you provide a shared access signature to a tenant, they can write their data to the blob. By using Event Grid integration for Azure Storage, your application can then be notified to process and import the data file.

Webhooks

Webhooks enable you to send events to your tenants at a URL that they provide to you. When you have information to send, you initiate a connection to the tenant's webhook and include your data in the HTTP request payload.

If you choose to build your own webhook eventing system, consider following the [CloudEvents](#) standard to simplify your tenants' integration requirements.

Alternatively, you can use a service like [Azure Event Grid](#) to provide webhook functionality. Event Grid works natively with CloudEvents, and supports [event domains](#), which are useful for multitenant solutions.

Note

Whenever you make outbound connections to your tenants' systems, remember that you're connecting to an external system. Follow recommended cloud practices, including using the **Retry pattern**, the **Circuit Breaker pattern**, and the **Bulkhead pattern** to ensure that problems in the tenant's system don't propagate to your system.

Delegated user access

When you access data from a tenant's data stores, consider whether you need to use a specific user's identity to access the data. When you do, your integration is subject to the same permissions that the user has. This approach is often called [delegated access](#).

For example, suppose your multitenant service runs machine learning models over your tenants' data. You need to access each tenant's instances of services, like Azure Synapse Analytics, Azure Storage, Azure Cosmos DB, and others. Each tenant has their own Azure AD instance. Your solution can be granted delegated access to the data store, so that you can retrieve the data that a specific user can access.

Delegated access is easier if the data store supports Azure AD authentication. [Many Azure services support Azure AD identities](#).

For example, suppose that your multitenant web application and background processes need to access Azure Storage by using your tenants' user identities from Azure AD. You might do the following steps:

1. [Create a multitenant Azure AD application registration](#) that represents your solution.
2. Grant the application [delegated permission to access Azure Storage as the signed-in user](#).
3. Configure your application to authenticate users by using Azure AD.

After a user signs in, Azure AD issues your application a short-lived access token that can be used to access Azure Storage on behalf of the user, and it issues a longer-lived refresh token. Your system needs to store the refresh token securely, so that your background processes can obtain new access tokens and can continue to access Azure Storage on behalf of the user.

Messaging

Messaging allows for asynchronous, loosely coupled communication between systems or components. Messaging is commonly used in integration scenarios to decouple the

source and destination systems. For more information on messaging and multitenancy, see [Architectural approaches for messaging in multitenant solutions](#).

When you use messaging as part of an integration with your tenants' systems, consider whether you should use [shared access signatures for Azure Service Bus](#) or [Azure Event Hubs](#). Shared access signatures enable you to grant limited access to your messaging resources to third parties, without enabling them to access the rest of your messaging subsystem.

In some scenarios, you might provide different service-level agreements (SLAs) or quality of service (QoS) guarantees to different tenants. For example, a subset of your tenants might expect to have their data export requests processed more quickly than others. By using the [Priority Queue pattern](#), you can create separate queues for different levels of priority, with different worker instances to prioritize them accordingly.

Composable integration components

Sometimes you might need to integrate with many different tenants, each of which uses different data formats or different types of network connectivity.

A common approach in integration is to build and test individual steps that perform the following types of actions:

- Retrieve data from a data store.
- Transform data to a specific format or schema.
- Transmit the data by using a particular network transport or to a known destination type.

Typically, you build these individual elements by using services like Azure Functions and Azure Logic Apps. You then define the overall integration process by using a tool like Logic Apps or Azure Data Factory, and it invokes each of the predefined steps.

When you work with complex multitenant integration scenarios, it can be helpful to define a library of reusable integration steps. Then, you can build workflows for each tenant to compose the applicable pieces together, based on that tenant's requirements. Alternatively, you might be able to expose some of the data sets or integration components directly to your tenants, so that they can build their own integration workflows from them.

Similarly, you might need to import data from tenants who use a different data format or different transport to others. A good approach for this scenario is to build tenant-specific *connectors*. Connectors are workflows that normalize and import the data into a

standardized format and location, and then they trigger your main shared import process.

If you need to build tenant-specific logic or code, consider following the [Anti-corruption Layer pattern](#). The pattern helps you to encapsulate tenant-specific components, while keeping the rest of your solution unaware of the added complexity.

If you use a [tiered pricing model](#), you might choose to require that tenants at a low pricing tier follow a standard approach with a limited set of data formats and transports. Higher pricing tiers might enable more customization or flexibility in the integration components that you offer.

Antipatterns to avoid

- **Exposing your primary data stores directly to tenants.** For example, avoid providing credentials to your data stores to your customers, and don't directly replicate data from your primary database to customers' read replicas of the same database system. Instead, create dedicated *integration data stores*, and use the [Valet Key pattern](#) to expose the data.
- **Exposing APIs without an API gateway.** APIs have specific concerns for access control, billing, and metering. Even if you don't plan to use API policies initially, it's a good idea to include an API gateway early. That way, if you need to customize your API policies in the future, you don't need to make breaking changes to the URLs that a third party depends on.
- **Unnecessary tight coupling.** Loose coupling, such as by using [messaging](#) approaches, can provide a range of benefits for security, performance isolation, and resiliency. When possible, it's a good idea to loosely couple your integrations with third parties. If you do need to tightly couple to a third party, ensure that you follow good practices like the [Retry pattern](#), the [Circuit Breaker pattern](#), and the [Bulkhead pattern](#).
- **Custom integrations for specific tenants.** Tenant-specific features or code can make your solution harder to test. It also makes it harder to modify your solution in the future, because you have to understand more code paths. Instead, try to build [composable components](#) that abstract the requirements for any specific tenant, and reuse them across multiple tenants with similar requirements.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributor:

- [Will Velida](#) | Customer Engineer 2, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [Architectural approaches for messaging in multitenant solutions](#).

Architectural approaches for IoT in a multitenant solution

Article • 03/20/2023

Multitenant IoT solutions come in many different flavors and sizes. You might have many requirements and constraints, ranging from infrastructure ownership, to customer data isolation, to compliance. It can be challenging to define a pattern that meets all of these design constraints, and doing so often requires considering multiple dimensions. This article describes several approaches commonly used to solve multitenancy considerations for IoT-based solutions. This document includes example multitenant architectures that leverage common components, according to the [IoT Reference Architecture](#).

Key considerations and requirements

These considerations and requirements are presented in the order in which they're typically prioritized for a solution's design.

Governance and compliance

Governance and compliance considerations might require that you use a particular pattern or set of IoT resources. Not all IoT services have the same certifications or capabilities. If you need to meet specific compliance standards, you might need to select specific services. Information on governance and compliance is covered in a [dedicated article on that topic](#).

Governance in IoT can also take additional forms, such as device ownership and management. Does the customer own the device or does the solution provider? Who owns the management of those devices? These considerations and implications are unique to each solution provider and can lead to different choices in the technology, deployment pattern, and multi-tenancy pattern that you use.

Scale

It's important to plan your solution's scale. Scale is often considered across these three dimensions:

- **Quantity of devices:** All Azure device management services - [Azure IoT Central](#), [Azure IoT Hub Device Provisioning Service \(DPS\)](#), and [Azure IoT Hub](#) - have

limitations on the number of devices supported in a single instance.

💡 Tip

Refer to the [high scale documentation](#), if you plan to deploy a very large number of devices.

- **Device throughput:** Different devices, even in the same solution, might have different throughput requirements. "Throughput" in this context refers to both the number of messages over a period of time and the size of the messages. For example, in a smart-building solution, thermostats will likely report data at a lower frequency than elevators, while in a connected-vehicle solution, vehicle camera recording data messages will likely be larger than navigation telemetry messages. If your messages are throttled with respect to frequency, you might need to scale out to more instances of a particular service, but if they are throttled with respect to size, you might need to scale up to larger instances of a particular service.
- **Tenants:** A single tenant's scale might be small, but when multiplied by the number of tenants, it can quickly grow.

Performance and reliability

Tenant isolation

Fully shared solutions can have [noisy neighbors](#). In the cases of IoT Hub and IoT Central, this can result in HTTP 429 ("Too Many Requests") response codes, which are hard failures that can cause a cascading effect. For more information, see [Quotas and Throttling](#).

In fully multitenant solutions, these effects can cascade. When customers share IoT Hubs or IoT Central applications, then all customers on the shared infrastructure will begin receiving errors. Because IoT Hub and IoT Central are commonly the entry points for data to the cloud, other downstream systems that depend on this data are likely to fail as well. Often, the most common occurrence for this to happen is when a message quota limit has been exceeded. In this situation, the fastest and simplest fix for IoT Hub solutions is to upgrade the IoT Hub SKU, increase the number of IoT Hub units, or both. For IoT Central solutions, the solution automatically scales as necessary, up to the [documented number of messages supported](#).

You can isolate and distribute tenants across the IoT control, management, and communications planes by using DPS's [custom allocation policies](#). Further, when you

follow the guidance for [high-scale IoT solutions](#), you can manage additional allocation distribution at the DPS load-balancer level.

Data storage, query, usage, and retention

IoT solutions tend to be very data-intensive, both when streaming and at rest. For more information on managing data in multitenant solutions, see [Architectural approaches for storage and data in multitenant solutions](#).

Approaches to consider

All the considerations that you'd normally make in an IoT architecture, for all the primary components (such as management, ingestion, processing, storage, security, and so on), are all choices you still must make when pursuing a multi-tenant solution. The primary difference is how you arrange and utilize the components to support multi-tenancy. For example, common decision points for storage might be deciding whether to use SQL Server or Azure Data Explorer, or perhaps on the ingestion and management tier, you'd choose between IoT Hub and IoT Central.

Most IoT solutions fit within a [root architecture pattern](#), which is a combination of the deployment target, tenancy model, and deployment pattern. These factors are determined by the key requirements and considerations described above.

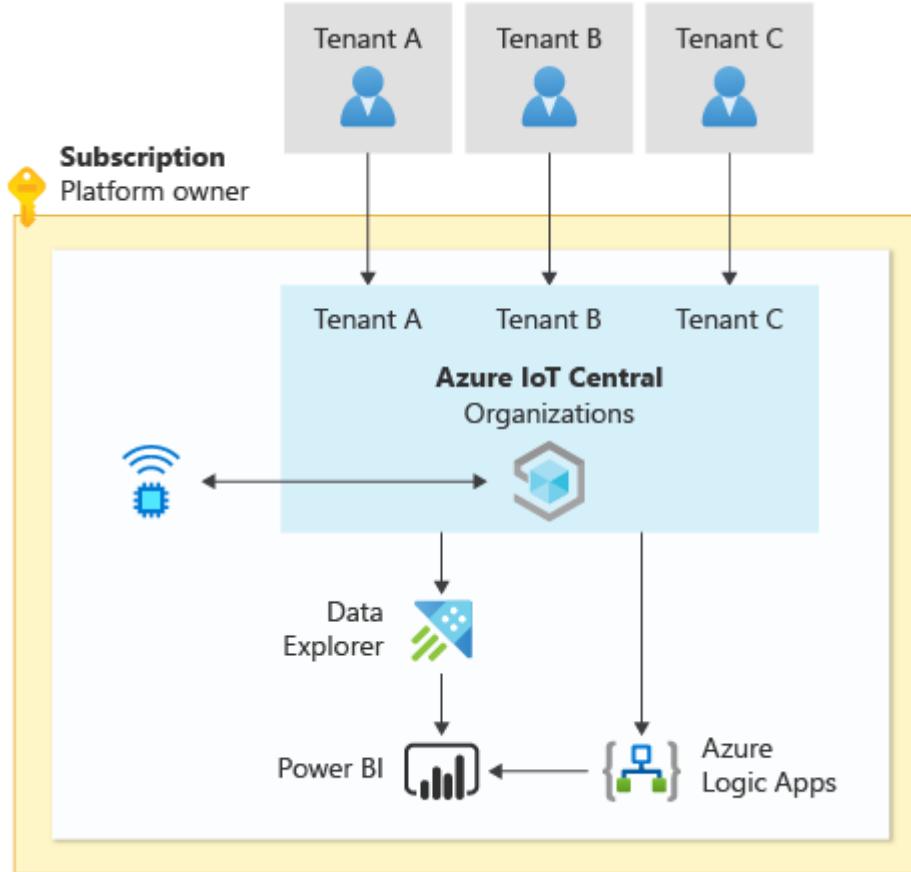
One of the largest decision points needing to be made, within the IoT space, is to select between an application-platform-as-a-service (aPaaS) and platform-as-a-service (PaaS) approaches. For more information, see [Compare Internet of Things \(IoT\) solution approaches \(PaaS vs. aPaaS\)](#).

This is the common "build vs. buy" dilemma that many organizations face in many projects. It's important to evaluate the advantages and disadvantages of both options.

Concepts and considerations for aPaaS solutions

A typical aPaaS solution using [Azure IoT Central](#), as the core of the solution, might use the following Azure PaaS and aPaaS services:

- [Azure Event Hubs](#) as a cross-platform, enterprise-grade messaging and data-flow engine.
- [Azure Logic Apps](#) as an integration platform-as-a-service, or iPaaS, offering.
- [Azure Data Explorer](#) as a data analytics platform.
- [Power BI](#) as a visualization and reporting platform.



In the previous diagram, the tenants share an IoT Central environment, Azure Data Explorer, Power BI, and Azure Logic Apps.

This approach is generally the fastest way to get a solution to market. It's a high scale service that supports multitenancy by using [organizations](#).

It's important to understand that because IoT Central is an aPaaS offering, there are certain decisions that are outside of an implementer's control. These decisions include:

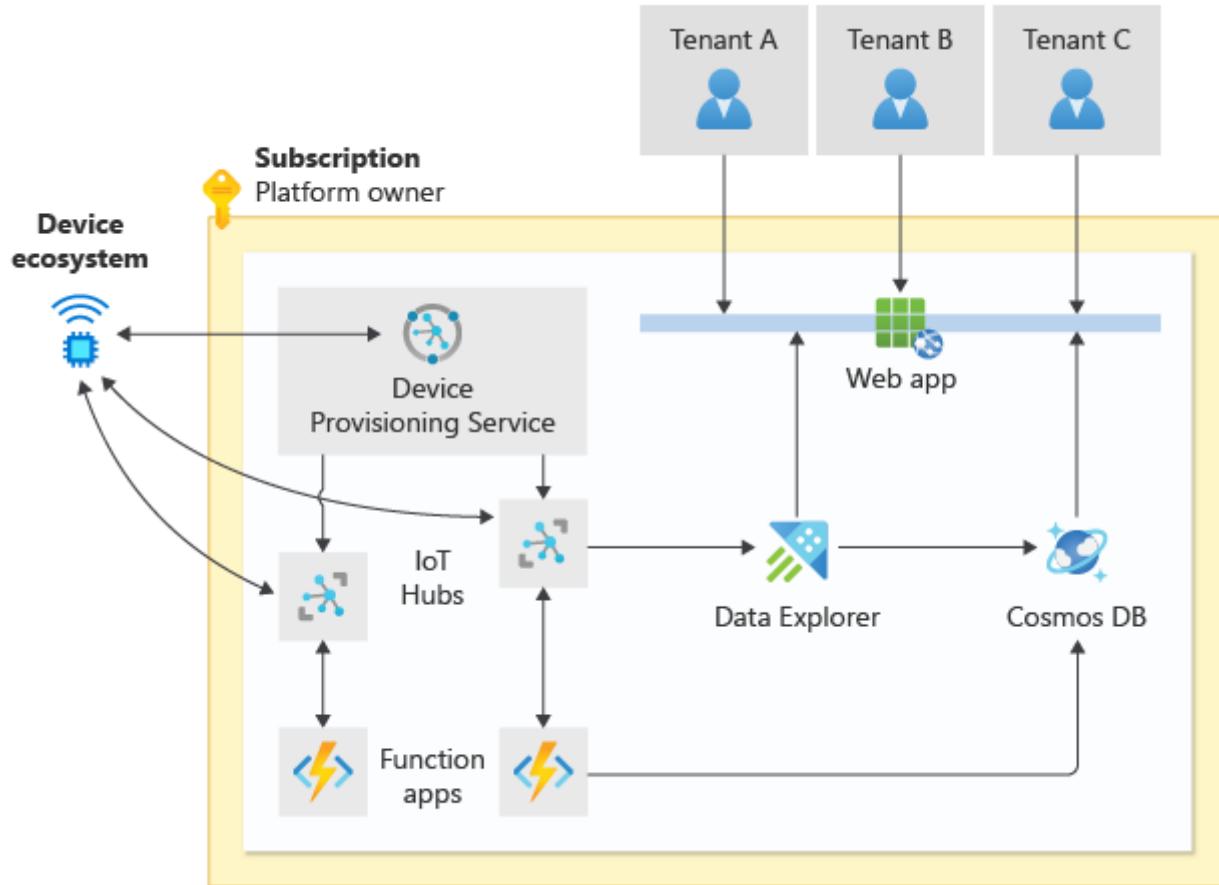
- IoT Central uses Azure Active Directory as its identity provider.
- IoT Central deployments are achieved using both control and data plane operations, which combine declarative documents with imperative code.
- In a multitenant pattern, both the IoT Central [maximum node limit](#) (which applies to both parents and leaves) and the maximum tree depth, might force a service provider to have multiple IoT Central instances. In that case, you should consider following the [Deployment Stamp pattern](#).
- IoT Central imposes [API call limits](#), which might impact large implementations.

Concepts and considerations for PaaS solutions

A PaaS-based approach might use the following Azure services:

- [Azure IoT Hub](#) as the core device configuration and communications platform.

- [Azure IoT Device Provisioning Service](#) as the device deployment and initial configuration platform.
- [Azure Data Explorer](#) for storing and analyzing warm and cold path time series data from IoT devices.
- [Azure Stream Analytics](#) for analyzing hot path data from IoT devices.
- [Azure IoT Edge](#) for running artificial intelligence (AI), third-party services, or your own business logic on IoT Edge devices.



In the previous diagram, each tenant connects to a shared web app, which receives data from IoT Hubs and a function app. Devices connect to the Device Provisioning Service and to IoT Hubs.

This approach requires more developer effort to create, deploy, and maintain the solution (versus an aPaaS approach). Fewer capabilities are prebuilt for the implementer's convenience. This means that this approach also offers more control, because fewer assumptions are embedded in the underlying platform.

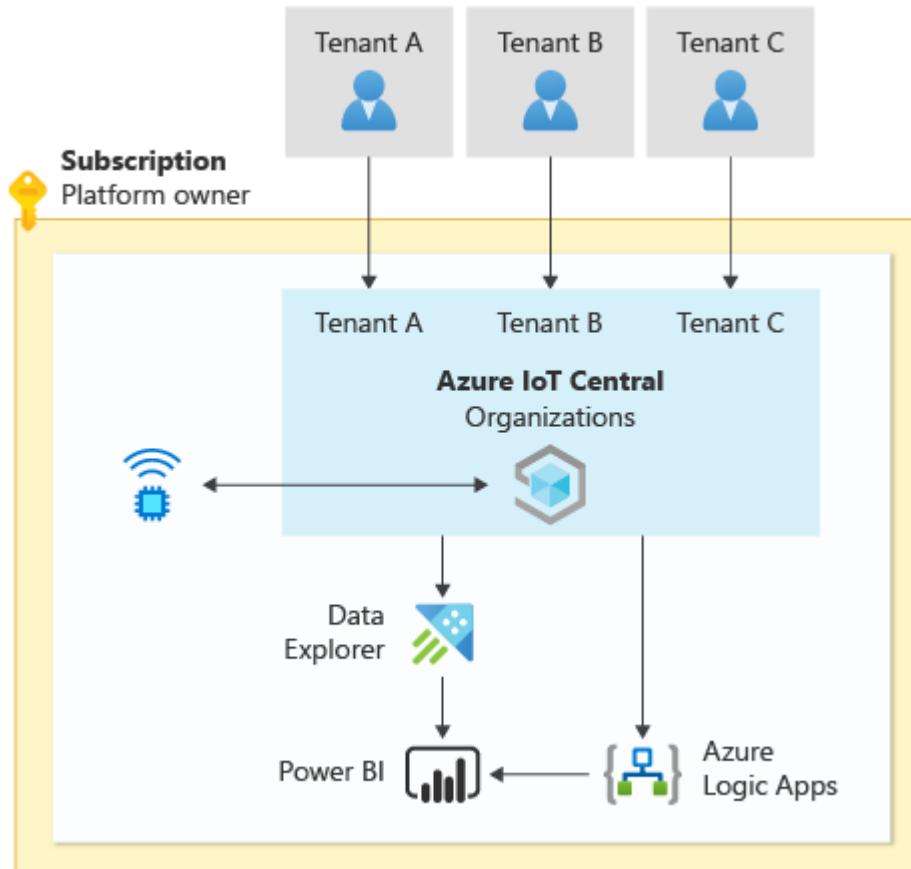
Root architecture patterns

The following table lists common patterns for multitenant IoT solutions. Each pattern includes the following information:

- The name of the **Pattern**, which is based on the combination of the target, model, and deployment type.
- The **Deployment target**, representing the Azure Subscription to deploy resources to.
- The **Tenancy model** being referenced by the pattern, as described at [Multitenancy models](#)
- The **Deployment pattern**, referring to a simple deployment with minimal deployment considerations, a [Geode pattern](#), or a [Deployment Stamp pattern](#).

Pattern	Deployment target	Tenancy model	Deployment pattern
Simple SaaS	Service provider's subscription	Fully multitenant	Simple
Horizontal SaaS	Service provider's subscription	Horizontally partitioned	Deployment Stamp
Single-tenant automated	Either service provider's or customer's subscription	Single tenant per customer	Simple

Simple SaaS



Deployment Target	Tenancy Model	Deployment Pattern
-------------------	---------------	--------------------

Deployment Target	Tenancy Model	Deployment Pattern
Service provider's subscription	Fully multitenant	Simple

The *Simple SaaS* approach is the simplest implementation for a SaaS IoT Solution. As the previous diagram shows, all of the infrastructure is shared, and the infrastructure has no geographic or scale stamping applied. Often, the infrastructure resides within a single Azure subscription.

Azure IoT Central supports the concept of [organizations](#). Organizations enable a solution provider to easily segregate tenants in a secure, hierarchical manner, while sharing the basic application design across all the tenants.

Communications to systems outside of IoT Central, such as for longer-term data analysis, along a cold path or connectivity with business operations, is done through other Microsoft PaaS and aPaaS offerings. These additional offerings might include the following services:

- [Azure Event Hubs](#) as a cross-platform, enterprise-grade messaging and data flow engine.
- [Azure Logic Apps](#) as an integration platform-as-a-service, or iPaaS.
- [Azure Data Explorer](#) as a data analytics platform.
- [Power BI](#) as a visualization and reporting platform.

If you compare the *Simple SaaS* approach with the [*Single tenant automated*](#) aPaaS model, many characteristics are similar. The primary difference between the two models is that in the *Single tenant automated* model, you deploy a distinct IoT Central instance for each tenant, while in the *Simple SaaS with aPaaS* model, you instead deploy a shared instance for multiple customers, and you create an IoT Central organization for each tenant.

As you're sharing a multitenanted data tier in this model, you'll need to implement row-level security, as described in [Architectural approaches for storage and data in multitenant solutions](#), in order to isolate the customer data.

Benefits:

- Easier to manage and operate relative to the other approaches presented here.

Risks:

- This approach might not easily [scale to very high numbers of devices, messages, or tenants](#).

- Because all of the services and components are shared, a failure in any component might affect all of your tenants. This is a risk to your solution's reliability and high availability.
- It's important to consider how you manage the compliance, operations, tenant lifecycle, and security of sub-fleets of devices. These considerations become important because of the shared nature of this solution type at the control, management, and communications planes.

Horizontal SaaS

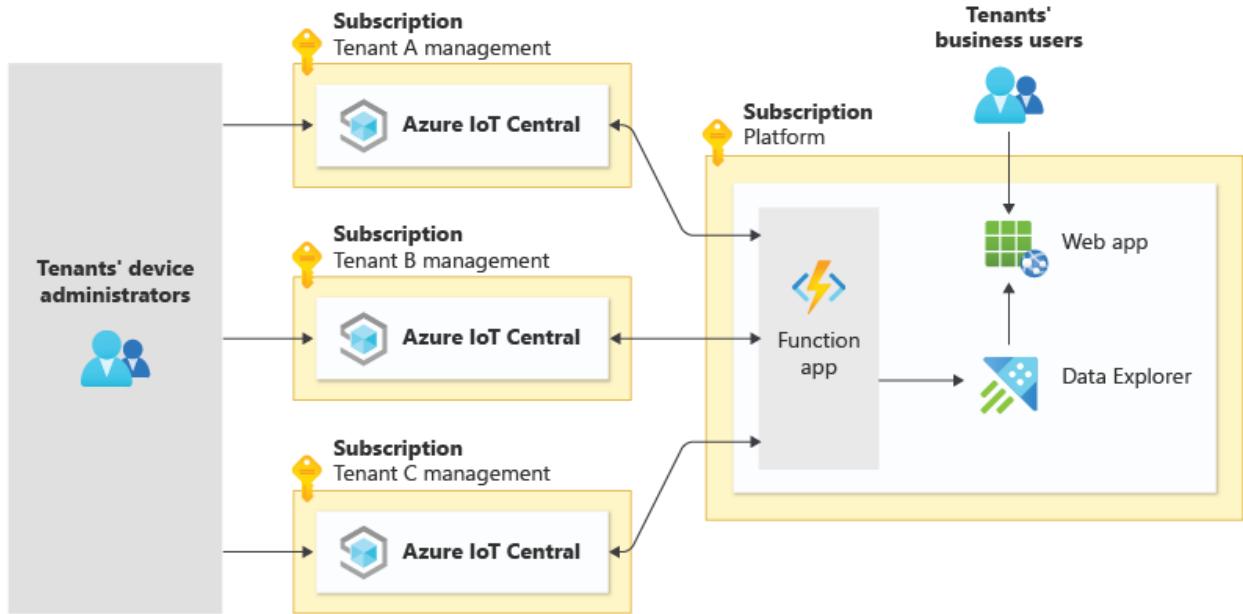
Deployment target	Tenancy model	Deployment pattern
Service provider's subscription	Horizontally partitioned	Deployment Stamp

A common scalability approach is to [horizontally partition the solution](#). This means you have some shared components and some per-customer components.

Within an IoT solution, there are many components that can be horizontally partitioned. The horizontally partitioned subsystems are typically arranged using a [deployment stamp pattern](#) which integrates with the greater solution.

Example horizontal SaaS

The below architectural example partitions IoT Central per end customer, which serves as the device management, device communications, and administrations portal. This is often done in such a way that the end customer who consumes the solution has full control over adding, removing, and updating devices themselves, without the intervention of the software vendor. The rest of the solution follows a standard shared infrastructure pattern, which solves for hot path analysis, business integrations, SaaS management, and device analysis needs.



Each tenant has their own IoT Central organization, which sends telemetry to a shared function app and makes it available to the tenants' business users through a web app.

Benefits:

- Generally easy to manage and operate, although additional management might be required for single-tenant components.
- Flexible scaling options, because layers are scaled as necessary.
- Impact of component failures is reduced. While a failure of a shared component impacts all customers, horizontally scaled components only impact the customers that are associated with specific scale instances.
- Improved per-tenant consumption insights for partitioned components.
- Partitioned components provide easier per-tenant customizations.

Risks:

- Consider the **scale** of the solution, especially for any shared components.
- Reliability and high availability are potentially impacted. A single failure in the shared components might affect all the tenants at once.
- The per-tenant partitioned component customization requires long-term DevOps and management considerations.

Below are the most common components that are typically suitable for horizontal partitioning.

Databases

You might choose to partition the databases. Often it's the telemetry and device data stores that are partitioned. Frequently, multiple data stores are used for different specific

purposes, such as warm versus archival storage, or for tenancy subscription status information.

Separate the databases for each tenant, for the following benefits:

- Support compliance standards. Each tenant's data is isolated across instances of the data store.
- Remediate noisy neighbor issues.

Device management, communications, and administration

Azure IoT Hub Device Provisioning Service, IoT Hub, and IoT Central applications can often be deployed as horizontally partitioned components. If you follow this approach, you need to have an additional service to redirect devices to the appropriate Device Provisioning Service for that particular tenant's management, control, and telemetry plane. For more information, see the whitepaper, [Scaling out an Azure IoT solution to support millions of devices](#).

This is often done to enable the end customers to manage and control their own fleets of devices that are more directly and fully isolated.

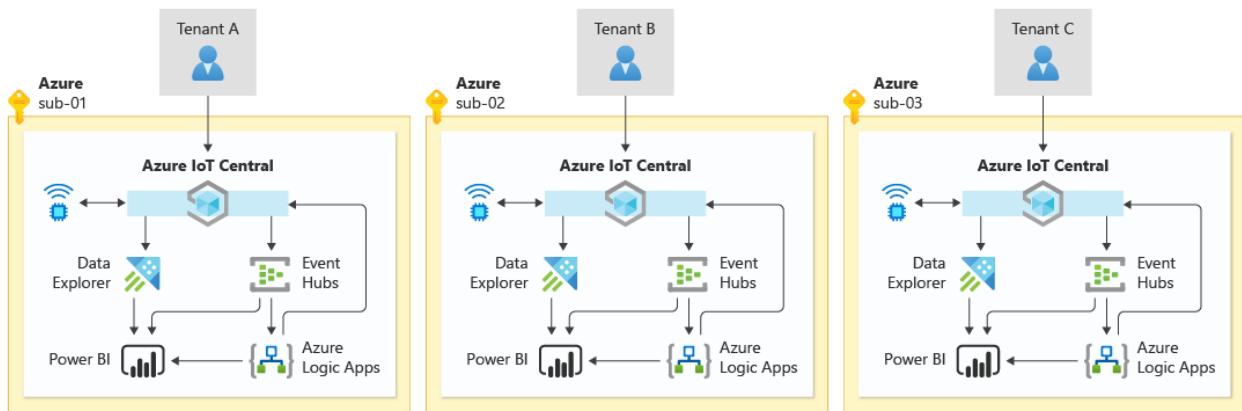
If the device communications plane is horizontally partitioned, telemetry data must be enriched with data for the source tenant, such that the stream processor knows which tenant rules to apply to the data stream. For example, if a telemetry message generates a notification in the stream processor, the stream processor will need to determine the proper notification path for the associated tenant.

Stream processing

By partitioning stream processing, you enable per-tenant customizations of the analysis within the stream processors.

Single-tenant automated

A single-tenant automated approach is based on a similar decision process and design to an [enterprise solution](#).



Each tenant has its own identical, isolated environment, with an IoT Central organization and other components dedicated to them.

Deployment Target	Tenancy Model	Deployment Pattern
Either service provider's or customer's subscription	Single tenant per customer	Simple

A critical decision point in this approach is choosing which Azure subscription the components should be deployed to. If the components are deployed to your subscription, you have more control and better visibility into the cost of the solution, but it requires you to own more of the solution's security and governance concerns. Conversely, if the solution is deployed in your customer's subscription, the customer is ultimately responsible for the security and governance of the deployment.

This pattern supports a high degree of scalability. This is because tenant and subscription requirements are generally the limiting factors in most solutions. Therefore, isolate each tenant to give a large scope for scaling each tenant's workload, without substantial effort on your part, as the solution developer.

This pattern also generally has low latency, when compared to other patterns, because you are able to deploy the solution components based on your customers' geography. Geographical affinity allows for shorter network paths between an IoT device and your Azure deployment.

If necessary, you can extend the automated deployment to support improved latency or scale, by allowing the quick deployment of extra instances of the solution, for a customer in existing or new geographies.

The *single-tenant automated* approach is similar to the *simple SaaS* aPaaS model. The primary difference between the two models is that in the *single-tenant automated* approach, you deploy a distinct IoT Central instance for each tenant, while in the *simple*

SaaS with aPaaS model, you deploy a shared instance of IoT Central with multiple IoT Central organizations.

Benefits:

- Relatively easy to manage and operate.
- Tenant isolation is essentially guaranteed.

Risks:

- Initial automation can be complicated for new development staff.
- Security of cross-customer credentials for higher-level deployment management must be enforced, or the compromises can extend across customers.
- Costs are expected to be higher, because the scale benefits of a shared infrastructure across customers are not available.
- If the solution provider owns the maintenance of each instance, you might have many instances to maintain.

Increase the scale of SaaS

When you expand the scale of a solution to very large deployments, there are specific challenges that arise based on service limits, geographic concerns, and other factors. For more information on large-scale IoT deployment architectures, see [Scaling out an Azure IoT solution to support millions of devices](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Michael C. Bazarewsky](#) | Senior Customer Engineer, FastTrack for Azure
- [David Crook](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirsingh](#) | Principal Customer Engineer, FastTrack for Azure

Next steps

- Review guidance for [multitenancy and Azure Cosmos DB](#).

- Learn about [hot, warm, and cold data paths with IoT on Azure](#).
- Refer to the [Azure IoT reference architectures](#).
- Review documentation on how to [Scale IoT solutions with deployment stamps](#).

Architectural approaches for AI and ML in multitenant solutions

Article • 05/10/2023

An ever-increasing number of multitenant solutions are built around artificial intelligence (AI) and machine learning (ML). A multitenant AI/ML solution is one that provides similar ML-based capabilities to any number of tenants. Tenants generally can't see or share the data of any other tenant, but in some situations, tenants might use the same models as other tenants.

Multitenant AI/ML architectures need to consider the requirements for data and models, as well as the compute resources that are required to train models and to perform inference from models. It's important to consider how multitenant AI/ML models are deployed, distributed, and orchestrated, and to ensure that your solution is accurate, reliable, and scalable.

Key considerations and requirements

When you work with AI and ML, it's important to separately consider your requirements for *training* and for *inference*. The purpose of training is to build a predictive model that's based on a set of data. You perform inference when you use the model to predict something in your application. Each of these processes has different requirements. In a multitenant solution, you should consider how your [tenancy model](#) affects each process. By considering each of these requirements, you can ensure that your solution provides accurate results, performs well under load, is cost-efficient, and can scale for your future growth.

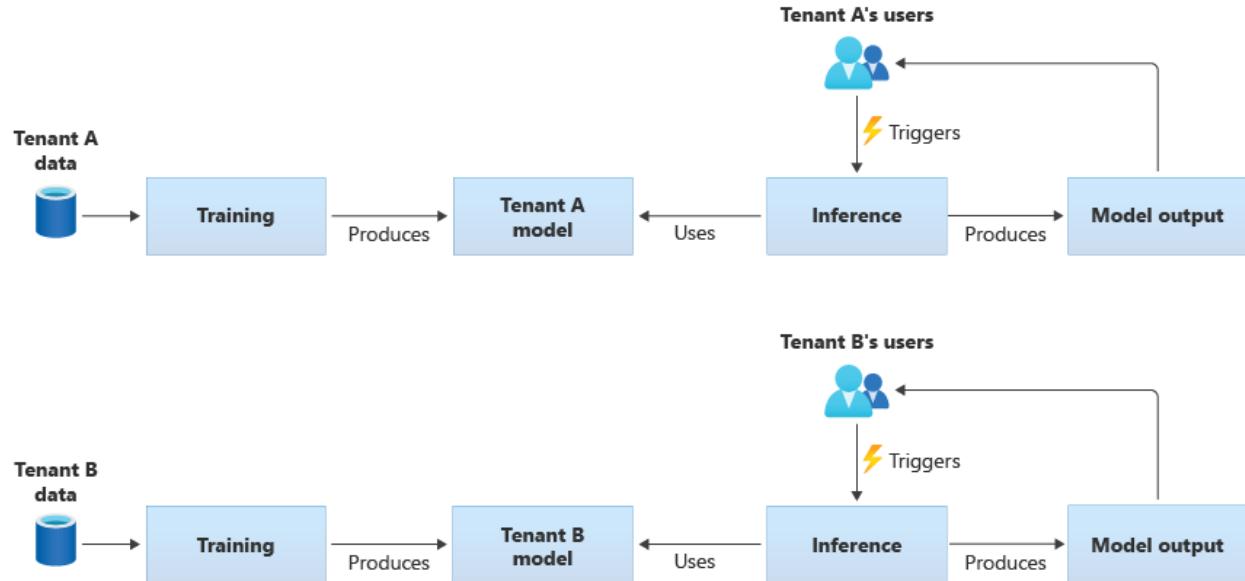
Tenant isolation

Ensure that tenants don't gain unauthorized or unwanted access to the data or models of other tenants. Treat models with a similar sensitivity to the raw data that trained them. Ensure that your tenants understand how their data is used to train models, and how models trained on other tenants' data might be used for inference purposes on their workloads.

There are three common approaches for working with ML models in multitenant solutions: tenant-specific models, shared models, and tuned shared models.

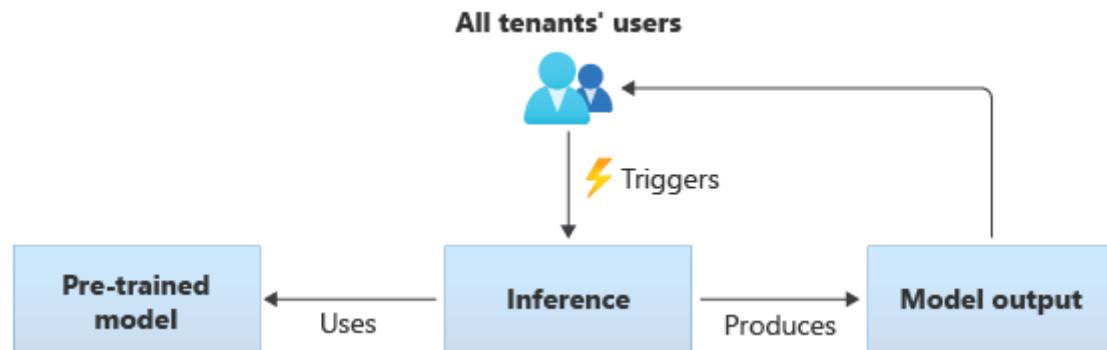
Tenant-specific models

Tenant-specific models are trained only on the data for a single tenant, and then they are applied to that single tenant. Tenant-specific models are appropriate when your tenants' data is sensitive, or when there's little scope to learn from the data that's provided by one tenant, and you apply the model to another tenant. The following diagram illustrates how you might build a solution with tenant-specific models for two tenants:

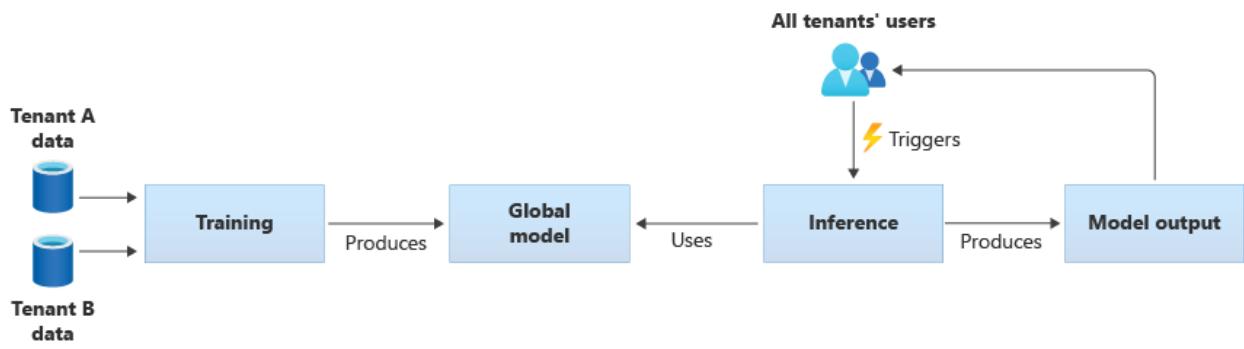


Shared models

In solutions that use shared models, all tenants perform inference based on the same shared model. Shared models might be pretrained models that you acquire or obtain from a community source. The following diagram illustrates how a single pretrained model can be used for inference by all tenants:



You also can build your own shared models by training them from the data provided by all of your tenants. The following diagram illustrates a single shared model, which is trained on data from all tenants:



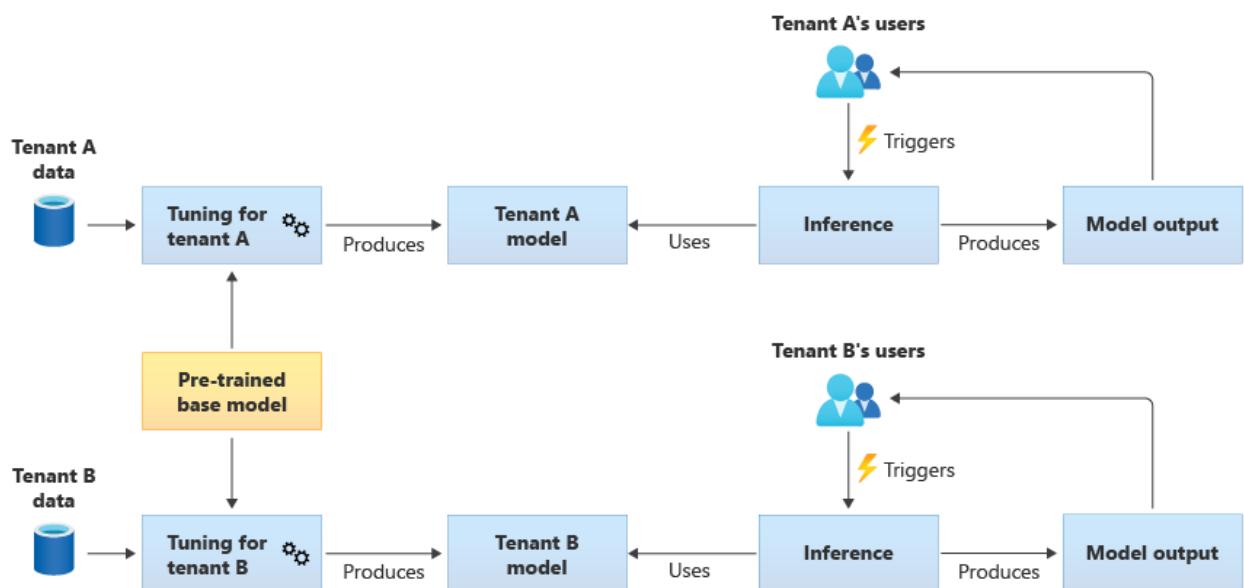
ⓘ Important

If you train a shared model from your tenants' data, ensure that your tenants understand and agree to the use of their data. Ensure identifying information is removed from your tenants' data.

Consider what to do, if a tenant objects to their data being used to train a model that will be applied to another tenant. For example, would you be able to exclude specific tenants' data from the training data set?

Tuned shared models

You also might choose to acquire a pretrained base model, and then perform further model tuning to make it applicable to each of your tenants, based on their own data. The following diagram illustrates this approach:



Scalability

Consider how the growth of your solution affects your use of AI and ML components. Growth can refer to an increase in the number of tenants, the amount of data stored for each tenant, the number of users, and the volume of requests to your solution.

Training: There are several factors that influence the resources that are required to train your models. These factors include the number of models you need to train, the amount of data that you train the models with, and the frequency at which you train or retrain models. If you create tenant-specific models, then as your number of tenants grows, the amount of compute resources and storage that you require will also be likely to grow. If you create shared models and train them based on data from all of your tenants, it's less likely that the resources for training will scale at the same rate as the growth in your number of tenants. However, an increase in the overall amount of training data will affect the resources that are consumed, to train both the shared and tenant-specific models.

Inference: The resources that are required for inference are usually proportional to the number of requests that access the models for inference. As the number of tenants increase, the number of requests is also likely to increase.

It's a good general practice to use Azure services that scale well. Because AI/ML workloads tend to make use of containers, Azure Kubernetes Service (AKS) and Azure Container Instances (ACI) tend to be common choices for AI/ML workloads. AKS is usually a good choice to enable high scale, and to dynamically scale your compute resources based on demand. For small workloads, ACI can be a simple compute platform to configure, although it doesn't scale as easily as AKS.

Performance

Consider the performance requirements for the AI/ML components of your solution, for both training and inference. It's important to clarify your latency and performance requirements for each process, so that you can measure and improve as required.

Training: Training is often performed as a batch process, which means that it might not be as performance-sensitive as other parts of your workload. However, you need to ensure that you provision sufficient resources to perform your model training efficiently, including as you scale.

Inference: Inference is a latency-sensitive process, often requiring a fast or even real-time response. Even if you don't need to perform inference in real time, ensure you monitor the performance of your solution and use the appropriate services to optimize your workload.

Consider using Azure's high-performance computing capabilities for your AI and ML workloads. Azure provides many different types of virtual machines and other hardware instances. Consider whether your solution would benefit from using CPUs, [GPUs](#), [FPGAs](#), or other hardware-accelerated environments. Azure also provides real-time inference with NVIDIA GPUs, including NVIDIA Triton Inference Servers. For low-priority compute requirements, consider using [AKS spot node pools](#). To learn more about optimizing compute services in a multitenant solution, see [Architectural approaches for compute in multitenant solutions](#).

Model training typically requires a lot of interactions with your data stores, so it's also important to consider your data strategy and the performance that your data tier provides. For more information about multitenancy and data services, see [Architectural approaches for storage and data in multitenant solutions](#).

Consider profiling your solution's performance. For example, [Azure Machine Learning provides profiling capabilities](#) that you can use when developing and instrumenting your solution.

Implementation complexity

When you build a solution to use AI and ML, you can choose to use prebuilt components, or to build custom components. There are two key decisions you need to make. The first is the *platform or service* you use for AI and ML. The second is whether you use pretrained models or build your own custom models.

Platforms: There are many Azure services that you can use for your AI and ML workloads. For example, Azure Cognitive Services and Azure OpenAI Service provide APIs to perform inference against prebuilt models, and Microsoft manages the underlying resources. Azure Cognitive Services enables you to quickly deploy a new solution, but it gives you less control over how training and inference are performed, and it might not suit every type of workload. In contrast, Azure Machine Learning is a platform that enables you to build, train, and use your own ML models. Azure Machine Learning provides control and flexibility, but it increases the complexity of your design and implementation. Review the [machine learning products and technologies from Microsoft](#) to make an informed decision when selecting an approach.

Models: Even when you don't use a full model that's provided by a service like Azure Cognitive Services, you can still accelerate your development by using a pretrained model. If a pretrained model doesn't precisely suit your needs, consider extending a pretrained model by applying a technique called *transfer learning* or *fine-tuning*. Transfer learning enables you to extend an existing model and apply it to a different domain. For example, if you're building a multitenant music recommendation service,

you might consider building off a pretrained model of music recommendations, and use transfer learning to train the model for a specific user's music preferences.

By using a prebuilt ML platforms like Azure Cognitive Services or Azure OpenAI Service, or a pretrained model, you can significantly reduce your initial research and development costs. The use of prebuilt platforms might save you many months of research, and avoid the need to recruit highly qualified data scientists to train, design, and optimize models.

Cost optimization

Generally, AI and ML workloads incur the greatest proportion of their costs from the compute resources that are required for model training and inference. Review [Architectural approaches for compute in multitenant solutions](#) to understand how to optimize the cost of your compute workload for your requirements.

Consider the following requirements when planning your AI and ML costs:

- **Determine compute SKUs for training.** For example, refer to [guidance on how to do this with Azure ML](#).
- **Determine compute SKUs for inference.** For an example cost estimate for inference, [refer to the guidance for Azure ML](#).
- **Monitor your utilization.** By observing the utilization of your compute resources, you can determine whether you should decrease or increase their capacity by deploying different SKUs, or scale the compute resources as your requirements change. See [Azure Machine Learning Monitor](#).
- **Optimize your compute clustering environment.** When you use compute clusters, monitor cluster utilization or configure [autoscaling](#) to scale down compute nodes.
- **Share your compute resources.** Consider whether you can optimize the cost of your compute resources by sharing them across multiple tenants.
- **Consider your budget.** Understand whether you have a fixed budget, and monitor your consumption accordingly. You can [set up budgets](#) to prevent overspending and to allocate quotas based on tenant priority.

Approaches and patterns to consider

Azure provides a set of services to enable AI and ML workloads. There are several common architectural approaches used in multitenant solutions: to use prebuilt AI/ML solutions, to build a custom AI/ML architecture by using Azure Machine Learning, and to use one of the Azure analytics platforms.

Use prebuilt AI/ML services

It's a good practice to try to use prebuilt AI/ML services, where you can. For example, your organization might be beginning to look at AI/ML and want to quickly integrate with a useful service. Or, you might have basic requirements that don't require custom ML model training and development. Prebuilt ML services enable you to use inference without building and training your own models.

Azure provides several services that provide AI and ML technology across a range of domains, including language understanding, speech recognition, knowledge, document and form recognition, and computer vision. Azure's prebuilt AI/ML services include [Azure Cognitive Services](#), [Azure OpenAI Service](#), [Azure Cognitive Search](#), and [Azure Form Recognizer](#). Each service provides a simple interface for integration, and a collection of pretrained and tested models. As managed services, they provide service-level agreements and require little configuration or ongoing management. You don't need to develop or test your own models to use these services.

Many managed ML services don't require model training or data, so there's usually no tenant data isolation concerns. However, when you work with Cognitive Search in a multitenant solution, review [Design patterns for multitenant SaaS applications](#) and [Azure Cognitive Search](#).

Consider the scale requirements for the components in your solution. For example, many of the APIs within Azure Cognitive Services support a maximum number of requests per second. If you deploy a single Cognitive Services resource to share across your tenants, then as the number of tenants increases, you might need to [scale to multiple resources](#).

Note

Some managed services enable you to train with your own data, including the [Custom Vision service](#), the [Face API](#), [Form Recognizer custom models](#), and some [OpenAI models that support customization and fine-tuning](#). When you work with these services, it's important to consider the [isolation requirements](#) for your tenants' data.

Custom AI/ML architecture

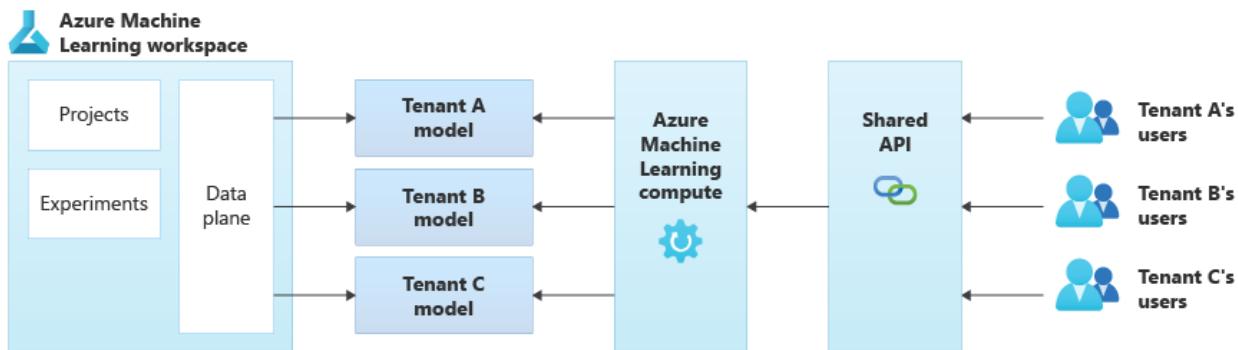
If your solution requires custom models, or you work in a domain that isn't covered by a managed ML service, then consider building your own AI/ML architecture. [Azure Machine Learning](#) provides a suite of capabilities to orchestrate the training and

deployment of ML models. Azure Machine Learning supports many open-source machine learning libraries, including [PyTorch](#), [Tensorflow](#), [Scikit](#), and [Keras](#). You can continuously monitor models' performance metrics, detect data drift, and trigger retraining to improve model performance. Throughout the lifecycle of your ML models, Azure Machine Learning enables auditability and governance with built-in tracking and lineage for all your ML artifacts.

When working in a multitenant solution, it's important to consider the [isolation requirements of your tenants](#) during both the training and inference stages. You also need to determine your model training and deployment process. Azure Machine Learning provides a pipeline to train models, and to deploy them to an environment to be used for inference. In a multitenant context, consider whether models should be deployed to shared compute resources, or if each tenant has dedicated resources. Design your model deployment pipelines, based on your [isolation model](#) and your [tenant deployment process](#).

When you use open-source models, you might need to retrain these models by using transfer learning or tuning. Consider how you will manage the different models and training data for each tenant, as well as versions of the model.

The following diagram illustrates an example architecture that uses Azure Machine Learning. The example uses the [tenant-specific models](#) isolation approach.



Integrated AI/ML solutions

Azure provides several powerful analytics platforms that can be used for a range of purposes. These platforms include [Azure Synapse Analytics](#), [Databricks](#), and [Apache Spark](#).

You can consider using these platforms for AI/ML, when you need to scale your ML capabilities to a very large number of tenants, and when you need large-scale compute and orchestration. You also might consider using these platforms for AI/ML, when you need a broad analytics platform for other parts of your solution, such as for data analytics and integration with reporting through Microsoft Power BI. You can deploy a

single platform that covers all of your analytics and AI/ML needs. When you implement data platforms in a multitenant solution, review [Architectural approaches for storage and data in multitenant solutions](#).

Antipatterns to avoid

- **Failure to consider isolation requirements.** It's important to carefully consider how you [isolate tenants' data and models](#), both for training and inference. Failing to do so might violate legal or contractual requirements. It also might reduce the accuracy of your models to train across multiple tenants' data, if the data is substantially different.
- **Noisy Neighbors.** Consider whether your training or inference processes could be subject to the [Noisy Neighbor problem](#). For example, if you have several large tenants and a single small tenant, ensure that the model training for the large tenants doesn't inadvertently consume all of the compute resources and starve the smaller tenants. Use resource governance and monitoring to mitigate the risk of a tenant's compute workload that's affected by the activity of the other tenants.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Kevin Ashley](#) | Senior Customer Engineer, FastTrack for Azure

Other contributors:

- [Paul Burpo](#) | Principal Customer Engineer, FastTrack for Azure
- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist
- [Arsen Vladimirsingh](#) | Principal Customer Engineer, FastTrack for Azure

Next steps

Review [Architectural approaches for compute in multitenant solutions](#) approaches.

Service-specific guidance for a multitenant solution

Article • 03/02/2023

When you're building a solution on Azure, you combine multiple distinct Azure services together to achieve your business and technical goals. Although Azure services work in a consistent manner, there are specific considerations for how you design and implement each service. When you design a multitenant solution, there are further considerations to review, for each service.

In this section, we provide guidance about the features of each service that are helpful for multitenant solutions. We also discuss the levels of tenant isolation that each service supports. Where applicable, we link to more details and sample implementations in the service's documentation.

ⓘ Note

The content in this section focuses specifically on the aspects of each service that relate to building a multitenant solution on Azure. For comprehensive information about each service and its features, refer to the service's documentation.

Intended audience

The content in this section is designed for architects, lead developers, and anyone building or implementing Azure components for a multitenant solution. The audience also includes independent software vendors (ISVs) and startups who develop SaaS solutions.

What's covered in this section?

The articles in this section describe some Azure services commonly used in multitenant solutions.

We frequently add new articles with guidance for additional services. You're also welcome to [submit suggestions for additional service-specific guidance ↗](#).

Next steps

Review the guidance for [Azure Resource Manager](#).

Multitenancy and Azure App Configuration

Article • 05/11/2023

[Azure App Configuration](#) enables you to store configuration settings for your application. By using Azure App Configuration, you can easily implement the [External Configuration Store pattern](#). In this article, we describe some of the features of Azure App Configuration that are useful when working with multitenanted systems, and we link to guidance and examples for how to use Azure App Configuration in a multitenant solution.

Isolation models

A *store* refers to a single instance of the Azure App Configuration service.

In a multitenant solution, it's common to have some settings that you share among multiple tenants, such as global settings or settings that apply to all tenants within a [deployment stamp](#). Global settings are often best stored within a shared App Configuration store. By following this approach, you minimize the number of places that you need to update when the value of a setting changes. This approach also minimizes the risk that settings could get out of sync.

Commonly, you will also have tenant-specific settings. For example, you might need to store each tenant's database name or internal identifiers. Or, you might want to specify different log levels for each tenant, such as when you diagnose a problem that's reported by a specific tenant and you need to collect diagnostic logs from that one tenant. You can choose whether to combine the tenant-specific settings for multiple tenants into a single store, or to deploy a store for each tenant. This decision should be based on your requirements. If your solution uses a single shared application tier for multiple tenants, there's likely to be minimal benefit to using tenant-specific stores. But if you deploy tenant-specific application instances, you might choose to mirror the same approach by deploying tenant-specific configuration stores.

The following table summarizes the differences between the main tenancy isolation models for Azure App Configuration:

Consideration	Shared store	Store per tenant
Data isolation	Low. Use key prefixes or labels to identify each tenant's data	High

Consideration	Shared store	Store per tenant
Performance isolation	Low	High
Deployment complexity	Low	Medium-high
Operational complexity	Low	Medium-high
Resource cost	Low	Medium-high
Example scenario	Large multitenant solution with a shared application tier	Premium tier tenants with fully isolated deployments

Shared stores

You can deploy a shared Azure App Configuration store for your whole solution, or for each stamp. You can then use the same store for all of your tenants' settings, and you can use [key prefixes](#) or [labels](#) to distinguish them.

If you need to store a large amount of data per tenant, or if you need to scale to a large number of tenants, you might be at risk of exceeding [any of the resource limits for a single store](#). In this scenario, consider whether you can shard your tenants across a set of shared stores, to minimize the deployment and management costs.

If you follow this approach, ensure you understand the [resource quotas and limits](#) that apply. In particular, be mindful of the total storage limit for the service tier you use, and ensure that you won't exceed the maximum requests per hour.

Stores per tenant

You might instead choose to deploy an Azure App Configuration store for each tenant. The Azure App Configuration [standard tier](#) enables you to deploy an unlimited number of stores in your subscription. However, this approach is often more complex to manage, because you have to deploy and configure more resources. There's also [a charge for each store resource that you deploy ↗](#).

Consider tenant-specific stores if you have one of the following situations:

- You need to use [customer-managed encryption keys](#), where the keys are separate for each tenant.
- Your tenants require their configuration data to be completely isolated from other tenants' data. Access permission for Azure App Configuration is controlled at the

store level, so by deploying separate stores, you can configure separate access permissions.

Features of Azure App Configuration that support multitenancy

When you use Azure App Configuration in a multitenant application, there are several features that you can use to store and retrieve tenant-specific settings.

Key prefixes

In Azure App Configuration, you work with key-value pairs that represent application settings. The key represents the name of the configuration setting. You can use a hierarchical naming structure for your keys. In a multitenant solution, consider using a tenant identifier as the prefix for your keys.

For example, suppose you need to store a setting to indicate the logging level for your application. In a single-tenant solution, you might name this setting `LogLevel`. In a multitenant solution, you might choose to use a hierarchical key name, such as `tenant1/LogLevel` for tenant 1, `tenant2/LogLevel` for tenant 2, and so on.

Azure App Configuration enables you to specify long key names, to support multiple levels in a hierarchy. If you choose to use long key names, ensure you understand the [size limits for keys and values](#).

When you load a single tenant's configuration into your application, you can specify a [key prefix filter](#) to only load that tenant's keys. You can also configure the provider library for Azure App Configuration to [trim the key prefix](#) from the keys, before it makes them available to your application. When you trim the key prefix, your application sees a consistent key name, with that tenant's values loaded into the application.

Labels

Azure App Configuration also supports [labels](#), which enable you to have separate values with the same key.

Labels are often used for versioning, working with multiple deployment environments, or for other purposes in your solution. While you can use tenant identifiers as labels, you won't be able to use labels for anything else. So, it's often a good practice to use [key prefixes](#) instead of labels, when you work with a multitenant solution.

If you do decide to use labels for each tenant, your application can load just the settings for a specific tenant by using a [label filter](#). This approach can be helpful if you have separate application deployments for each tenant.

Application-side caching

When you work with Azure App Configuration, it's important to cache the settings within your application, instead of loading them every time you use them. The [Azure App Configuration provider libraries](#) cache settings and refresh them automatically.

You also need to decide whether your application loads the settings for a single tenant or for all tenants.

As your tenant base grows, the amount of time and the memory required to load settings for all tenants together is likely to increase. So, in most situations, it's a good practice to load the settings for each tenant separately, when your application needs them.

If you load each tenant's configuration settings separately, your application needs to cache each set of settings separately to any others. In .NET applications, consider using an [in-memory cache](#) to cache the tenant's `IConfiguration` object and then use the tenant identifier as the cache key. By using an in-memory cache, you don't need to reload a configuration upon every request, but the cache can remove unused instances if your application is under memory pressure. You can also configure expiration times for each tenant's configuration settings.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [Zhenlan Wang](#) | Principal Software Engineering Manager, Azure App Configuration

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [deployment and configuration approaches for multitenancy](#).

Multitenancy and Azure Resource Manager

Article • 03/30/2023

Azure Resource Manager is the core resource management service for Azure. Every resource in Azure is created, managed, and eventually deleted through Resource Manager. When you build a multitenant solution, you often work with Resource Manager to dynamically provision resources for each tenant. On this page, we describe some of the features of Resource Manager that are relevant to multitenant solutions. We'll also provide links to guidance that can help you when you're planning to use Resource Manager.

Features of Resource Manager that support multitenancy

Infrastructure as code

Resource Manager provides tooling to support infrastructure as code, sometimes referred to as IaC. Infrastructure as code is important for all solutions in the cloud, but when working with multitenant solutions, it becomes particularly important. A multitenant solution often requires you to scale deployments, and to provision new resources as you onboard new tenants. If you manually create or configure resources, then you introduce extra risk and time to the process. It results in a less reliable deployment process overall.

When deploying your infrastructure as code from a deployment pipeline, we recommend you use [Bicep](#), which is a language specifically designed to deploy and manage Azure resources in a declarative way. You can also use [JSON Azure Resource Manager templates](#) (ARM templates), Terraform, or other third-party products that access the underlying Resource Manager APIs.

[Template specs](#) can be useful for provisioning new resources, deployment stamps, or environments from a single and well-parameterized template. By using template specs, you can create a central repository of the templates that you use to deploy your tenant-specific infrastructure. The templates are stored and managed within Azure itself, and you can reuse the template specs whenever you need to deploy from them.

In some solutions, you might choose to write custom code to dynamically provision or configure resources, or to initiate a template deployment. The [Azure SDKs](#) can be

used from your own code, to manage your Azure environment. Ensure that you follow good practices around managing the authentication of your application to Resource Manager, and use [managed identities](#) to avoid storing and managing credentials.

Role-based access control

[Role-based access control](#) (Azure RBAC) provides you with a fine-grained approach to manage access to your Azure resources. In a multitenant solution, consider whether you have resources that should have specific Azure RBAC policies applied. For example, you might have some tenants with particularly sensitive data, and you might need to apply RBAC to grant access to certain individuals, without including other people in your organization. Similarly, tenants might ask to access their Azure resources directly, such as during an audit. Should you choose to allow this, finely scoped RBAC permissions can enable you to grant access to a tenant's data, without providing access to other tenants' data.

Tags

[Tags](#) enable you to add custom metadata to your Azure resources, resource groups, and subscriptions. Consider tagging your tenant-specific resources with the tenant's identifier so that you can easily [track and allocate your Azure costs](#), and to simplify your resource management.

Azure resource quotas

Resource Manager is one of the points in Azure that enforces [limits and quotas](#). These quotas are important to consider throughout your design process. All Azure resources have limits that need to be adhered to, and these limits include the number of requests that can be made against Resource Manager within a certain time period. If you exceed this limit, [Resource Manager throttles the requests](#).

When you build a multitenant solution that performs automated deployments, you might reach these limits faster than other customers. Similarly, multitenant solutions that provision large amounts of infrastructure can trigger the limits.

Every Azure service is managed by a *resource provider*, which may also define its own limits. For example, the Azure Compute Resource Provider manages the provisioning of virtual machines, and [it defines limits on the number of requests](#) that can be made in a short period. Some other resource provider limits are documented in [Resource provider limits](#).

If you are at risk of exceeding the limits defined by Resource Manager or a resource provider, consider the following mitigations:

- Shard your workload across multiple Azure subscriptions.
- Use multiple resource groups within subscriptions.
- Send requests from different Azure Active Directory (Azure AD) principals.
- Request additional quota allocations. In general, quota allocation requests are submitted by opening a support case, although some services provide APIs for these requests, such as for [virtual machine reserved instances](#).

The mitigations you select need to be appropriate for the specific limit you encounter.

Isolation models

In some multitenant solutions, you might decide to deploy separate or dedicated resources for each tenant. Resource Manager provides several models that you can use to isolate resources, depending on your requirements and the reason you choose to isolate the resources. See [Azure resource organization in multitenant solutions](#) for guidance about how to isolate your Azure resources.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributor:

- [Arsen Vladimirsingh](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [deployment and configuration approaches for multitenancy](#).

Azure App Service and Azure Functions considerations for multitenancy

Azure Azure App Service Azure Functions

Azure App Service is a powerful web application hosting platform. Azure Functions, built on top of the App Service infrastructure, enables you to easily build serverless and event-driven compute workloads. Both services are frequently used in multitenant solutions.

Features of Azure App Service and Azure Functions that support multitenancy

Azure App Service and Azure Functions include many features that support multitenancy.

Custom domain names

Azure App Service enables you to use [wildcard DNS](#) and to add your own [wildcard TLS certificates](#). When you use [tenant-specific subdomains](#), wildcard DNS and TLS certificates enable you to easily scale your solution to a large number of tenants, without requiring a manual reconfiguration for each new tenant.

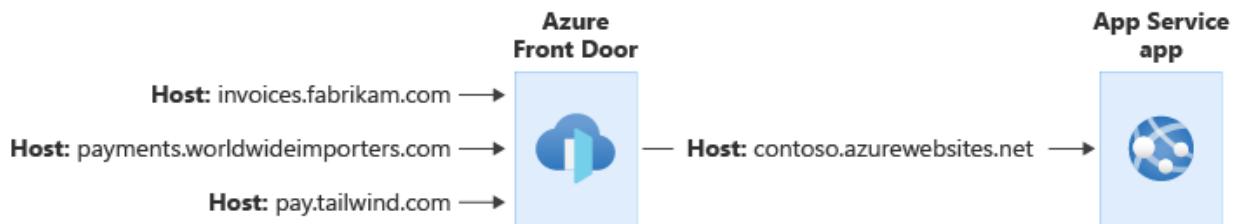
When you use [tenant-specific custom domain names](#), you might have a large number of custom domain names that need to be added to your app. It can become cumbersome to manage a lot of custom domain names, especially when they require individual TLS certificates. App Service provides [managed TLS certificates](#), which reduces the work required when you work with custom domains. However, there are [limits to consider](#), such as how many custom domains can be applied to a single app.

Integration with Azure Front Door

App Service and Azure Functions integrate with [Azure Front Door](#), to act as the internet-facing component of your solution. Azure Front Door enables you to add a web application firewall (WAF) and edge caching, and it provides other performance

optimizations. You can easily reconfigure your traffic flows to direct traffic to different backends, based on changing business or technical requirements.

When you use Azure Front Door with a multitenant app, you can use it to manage your custom domain names and to terminate your TLS connections. Your App Service application is then configured with a single hostname, and all traffic flows through to that application, which helps you avoid managing custom domain names in multiple places.



As in the above example, [Azure Front Door can be configured to modify the request's Host header](#). The original `Host` header sent by the client is propagated through the `x-Forwarded-Host` header, and your application code can use this header to [map the request to the correct tenant](#).

⚠️ Warning

If your application sends cookies or redirection responses, you need to take special care. Changes in the request's `Host` headers might invalidate these responses. For more information, see the [host name preservation best practice](#).

You can use [private endpoints](#) or App Service [access restrictions](#) to ensure that traffic has flowed through Front Door before reaching your app.

For more information about using Azure Front Door in a multitenant solution, see [Use Azure Front Door in a multitenant solution](#)

Authentication and authorization

Azure App Service can [validate authentication tokens on behalf of your app](#). When App Service receives a request, it checks to see whether each of the following conditions are met:

- The request contains a token.
- The token is valid.
- The request is authorized.

If any of the conditions aren't met, App Service can block the request, or it can redirect the user to your identity provider so that they can sign in.

If your tenants use Microsoft Entra ID as their identity system, you can configure Azure App Service to use [the /common endpoint](#) to validate user tokens. This ensures that, regardless of the user's Microsoft Entra tenant, their tokens are validated and accepted.

You can also integrate Azure App Service with Azure AD B2C for authentication of consumers.

More information:

- [App Service authorization](#)
- [Configure authentication in a sample web app by using Azure AD B2C](#)
- [Working with multitenant Microsoft Entra identities](#)

Access restrictions

You can restrict the traffic to your app by using [access restrictions](#). These can be used to specify the IP address ranges or the virtual networks that are allowed to connect to the app.

When you work with a multitenant solution, be aware of the maximum number of access restriction rules. For example, if you need to create an access restriction rule for every tenant, you might exceed the maximum number of rules that are allowed. If you need a larger number of rules, consider deploying a reverse proxy like [Azure Front Door](#).

Isolation models

When working with a multitenant system using Azure App Service or Azure Functions, you need to make a decision about the level of isolation that you want to use. Refer to the [tenancy models to consider for a multitenant solution](#) and to the guidance provided in the [architectural approaches for compute in multitenant solutions](#), to help you select the best isolation model for your scenario.

When you work with Azure App Service and Azure Functions, you should be aware of the following key concepts:

- In Azure App Service, a [plan](#) represents your hosting infrastructure. An app represents a single application running on that infrastructure. You can deploy multiple apps to a single plan.
- In Azure Functions, your hosting and application are also separated, but you have [additional hosting options available](#) for *elastic hosting*, where Azure Functions

manages scaling for you. For simplicity, we refer to the hosting infrastructure as a *plan* throughout this article, because the principles described here apply to both App Service and Azure Functions, regardless of the hosting model you use.

The following table summarizes the differences between the main tenancy isolation models for Azure App Service and Azure Functions:

[\[+\] Expand table](#)

Consideration	Shared apps	Apps per tenant with shared plans	Plans per tenant
Configuration isolation	Low	Medium. App-level settings are dedicated to the tenant, plan-level settings are shared	High. Each tenant can have their own configuration
Performance isolation	Low	Low-medium. Potentially subject to noisy neighbor issues	High
Deployment complexity	Low	Medium	High
Operational complexity	Low	High	High
Resource cost	Low	Low-high depending on application	High
Example scenario	Large multitenant solution with a shared application tier	Migrate applications that aren't aware of tenancy into Azure while gaining some cost efficiency	Single-tenant application tier

Shared apps

You might deploy a shared application on a single plan, and use the shared application for all your tenants.

This tends to be the most cost-efficient option, and it requires the least operational overhead because there are fewer resources to manage. You can scale the overall plan based on load or demand, and all tenants sharing the plan will benefit from the increased capacity.

It's important to be aware of the [App Service quotas and limits](#), such as the maximum number of custom domains that can be added to a single app, and the maximum number of instances of a plan that can be provisioned.

To be able to use this model, your application code must be multitenancy-aware.

Apps per tenant with shared plans

You can also choose to share your plan between multiple tenants, but deploy separate apps for each tenant. This provides you with logical isolation between each tenant, and this approach gives you the following advantages:

- **Cost efficiency:** By sharing your hosting infrastructure, you can generally reduce your overall costs per tenant.
- **Separation of configuration:** Each tenant's app can have its own domain name, TLS certificate, access restrictions, and token authorization policies applied.
- **Separation of upgrades:** Each tenant's application binaries can be upgraded independently of other apps on the same plan.

However, because the plan's compute resources are shared, the apps might be subject to the [Noisy Neighbor problem](#). Additionally, there are [limits to how many apps can be deployed to a single plan](#).

ⓘ Note

Don't use **deployment slots** for different tenants. Slots don't provide resource isolation. They are designed for deployment scenarios when you need to have multiple versions of your app running for a short time, such as blue-green deployments and a canary rollout strategy.

Plans per tenant

The strongest level of isolation is to deploy a dedicated plan for a tenant. This dedicated plan ensures that the tenant has full use of all of the server resources that are allocated to that plan.

This approach enables you to scale your solution to provide performance isolation for each tenant, and to avoid the [Noisy Neighbor problem](#). However, it also has a higher cost because the resources aren't shared with multiple tenants. Also, you need to consider the [maximum number of plans](#) that can be deployed into a single Azure resource group.

Host APIs

You can host APIs on both Azure App Service and Azure Functions. Your choice of platform will depend on the specific feature set and scaling options you need.

Whichever platform you use to host your API, consider using [Azure API Management](#) in front of your API application. API Management provides many features that can be helpful for multitenant solutions, including the following:

- A centralized point for all [authentication](#). This might include determining the tenant identifier from a token claim or other request metadata.
- [Routing requests to different API backends](#), which might be based on the request's tenant identifier. This can be helpful when you host multiple [deployment stamps](#), with their own independent API applications, but you need to have a single API URL for all requests.

Networking and multitenancy

IP addresses

Many multitenant applications need to connect to tenants' on-premises networks to send data.

If you need to send outbound traffic from a known static IP address or from a set of known static IP addresses, consider using a NAT Gateway. For more information about how to use NAT Gateway in multitenant solutions, see [Azure NAT Gateway considerations for multitenancy](#). App Service provides [guidance on how to integrate with a NAT Gateway](#).

If you don't need a static outbound IP address, but instead you need to occasionally check the IP address that your application uses for outbound traffic, you can [query the current IP addresses of the App Service deployment](#).

Quotas

Because App Service is itself a multitenant service, you need to take care about how you use shared resources. Networking is an area that you need to pay particular attention to, because there are [limits](#) that affect how your application can work with both inbound and outbound network connections, including source network address translation (SNAT) and TCP port limits.

If your application connects to a large number of databases or external services, then your app might be at risk of [SNAT port exhaustion](#). In general, SNAT port exhaustion indicates that your code isn't correctly reusing TCP connections, and even in a multitenant solution, you should ensure you follow the recommended practices for reusing connections.

However, in some multitenant solutions, the number of outbound connections to distinct IP addresses can result in SNAT port exhaustion, even when you follow good coding practices. In these scenarios, consider one of the following options:

- Deploy NAT Gateway to increase the number of SNAT ports that are available for your application to use. For more information about how to use NAT Gateway in multitenant solutions, see [Azure NAT Gateway considerations for multitenancy](#).
- Use [service endpoints](#) when you connect to Azure services, to bypass load balancer limits.

Even with these controls in place, you might approach limits with a large number of tenants, so you should plan to scale to additional App Service plans or [deployment stamps](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Thiago Almeida](#) | Principal Program Manager, Azure Functions
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [Resources](#) for architects and developers of multitenant solutions.

Considerations for using Container Apps in a multitenant solution

Article • 12/16/2022

You can use Azure Container Apps to run microservices and containerized applications on a serverless platform. This article describes some of the features of Container Apps that are useful for multitenant solutions. It also provides links to guidance that can help you during your planning phase.

Isolation models

When you work with a multitenant system that uses Container Apps, you need to determine the required level of isolation. Container Apps supports different models of multitenancy:

- You can implement *trusted multitenancy* by using a shared environment. For example, this model might be appropriate when your tenants are all from within your organization.
- You can implement *hostile multitenancy* by deploying separate environments for each tenant. For example, this model might be appropriate when you don't trust the code that your tenants run.

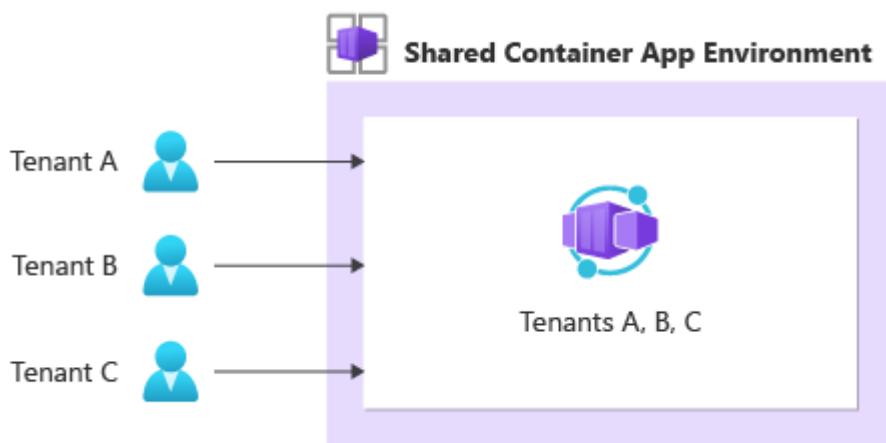
The following table summarizes the differences between the main tenancy isolation models for Container Apps. The models are described later in this article.

Consideration	One environment per tenant	Tenant-specific container apps	Shared container apps
Data isolation	High	Low	Low
Performance isolation	High	Medium. No network isolation.	Low
Deployment complexity	Medium	Low to medium	Low
Operational complexity	Medium	Low	Low
Resource cost	High	Low	Low

Consideration	One environment per tenant	Tenant-specific container apps	Shared container apps
Example scenario	Running hostile multitenant workloads in isolated environments for security and compliance	Optimizing cost, networking resources, and operations for trusted multitenant applications	Implementing a multitenant solution at the business-logic level

Shared container apps

You might want to consider deploying shared container apps in a single Container Apps environment that's used for all your tenants.



This approach is generally cost-efficient, and it requires the least operational overhead because there are fewer resources to manage.

However, if you want to use this isolation model, your application code must be multitenancy-aware. This isolation model doesn't guarantee isolation at the network, compute, monitoring, or data level. Your application code must handle tenant isolation. This model isn't appropriate for hostile multitenancy workloads in which you don't trust the code that's running.

Also, this model is potentially subject to [noisy neighbor concerns](#): one tenant's workload might affect the performance of another tenant's workload. If you need to provide dedicated throughput to mitigate this concern, the shared container apps model might not be appropriate.

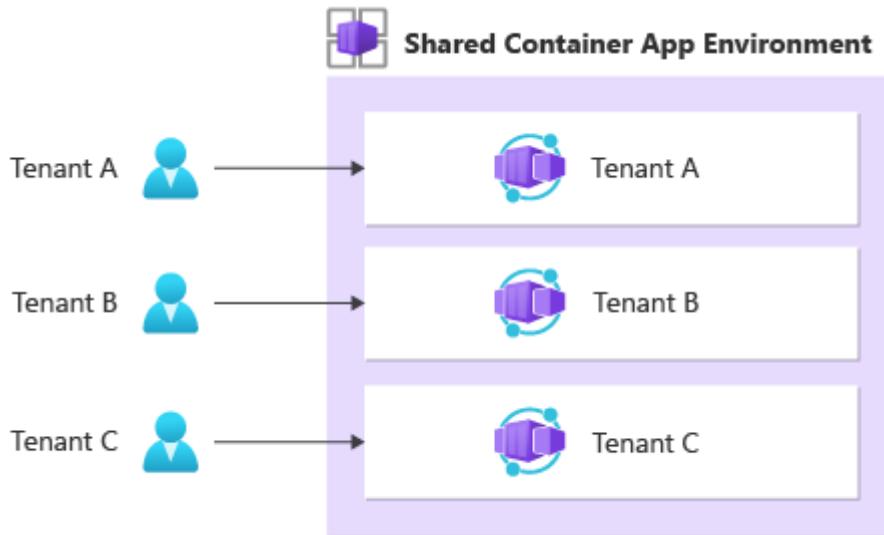
ⓘ Note

The [Deployment Stamps pattern](#) is useful when tenants are on different costing models. For example, tenants might be assigned to shared or dedicated Container Apps environments depending on their pricing tier. This deployment strategy

allows you to go beyond the limits of Container Apps for a single subscription per region and scale linearly as the number of tenants grows.

Tenant-specific container apps

Another approach that you might consider is isolating your tenants by deploying tenant-specific container apps within a shared environment.



This isolation model provides logical isolation between each tenant. It provides these advantages:

- **Cost efficiency.** By sharing a Container Apps environment, virtual network, and other attached resources like a Log Analytics workspace, you can generally reduce your overall cost and management complexity per tenant.
- **Separation of upgrades and deployments.** Each tenant's application binaries can be deployed and upgraded independently from those of other container apps in the same environment. This approach can be helpful if you need to upgrade different tenants to specific versions of your code at different times.
- **Resource isolation.** Each container app within your environment is allocated its own CPU and memory resources. If a specific tenant requires more resources, you can allocate more CPU and memory to that tenant's specific container app. Keep in mind that there are [limits on total CPU and memory allocations](#) on container apps.

However, this approach provides no hardware or network isolation between tenants. All container apps in a single environment share the same virtual network. You need to be able to trust that the workloads deployed to the apps won't misuse the shared resources.

There are also [limits on how many container apps you can deploy into a single environment](#). Take into account the expected growth in the number of tenants before

you implement this isolation model.

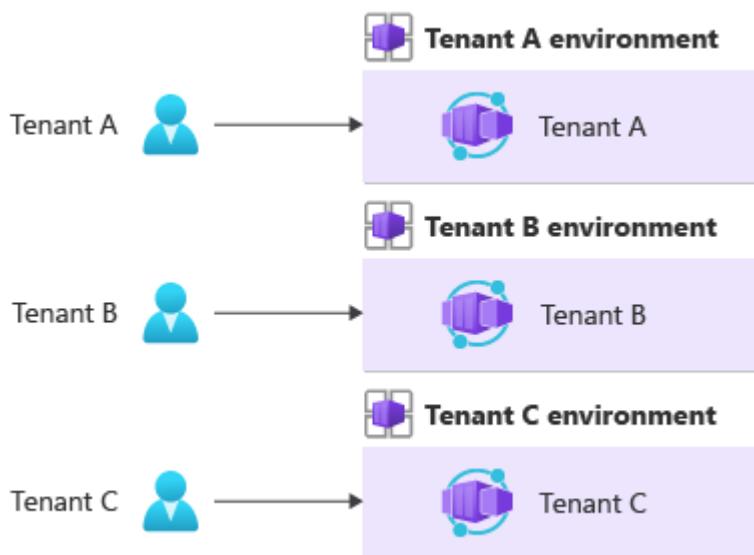
Container Apps has built-in support for Dapr, which uses a modular design to deliver functionality as [components](#). In Container Apps, Dapr components are environment-level resources. When you share a single environment across multiple tenants, ensure that you properly scope the Dapr components to the correct tenant-specific container app to guarantee isolation and avoid the risk of data leakage.

ⓘ Note

Don't use [revisions](#) to create different versions of your app for different tenants. Revisions don't provide resource isolation. They're designed for deployment scenarios in which you need to have multiple versions of your app running as part of an update rollout process, as in blue/green deployments and A/B testing.

One environment per tenant

You might consider deploying one Container Apps environment for each of your tenants. A [Container Apps environment](#) is the isolation boundary around a group of container apps. An environment provides compute and network isolation on the data plane. Each environment is deployed into its own virtual network, which is shared by all apps within the environment. Each environment has its own Dapr and monitoring configuration.



This approach provides the strongest level of data and performance isolation because each tenant's data and traffic are isolated to a specific environment. And when you use this model, your applications don't need to be multitenancy-aware. When you use this approach, you have more granular control over how you allocate resources to container

apps within the environment. You can determine allocations based on your tenant's requirements. For example, some tenants might require more CPU and memory resources than others, so you can provide more resources to these tenants' applications while benefiting from the isolation that tenant-specific environments provide.

However, there are low [limits on how many environments you can deploy within a subscription per region](#). In some situations, you can increase these quotas by creating an [Azure support ticket ↗](#).

Be sure to know the expected growth in the number of tenants before you implement this isolation model. Keep in mind that this approach often incurs a higher total cost of ownership, and higher levels of deployment and operational complexity, because of the extra resources you need to deploy and manage.

Container Apps features that support multitenancy

Custom domain names

Container Apps enables you to use [wildcard DNS](#) and to add your own [wildcard TLS certificates](#). When you use tenant-specific subdomains, wildcard DNS and TLS certificates enable you to easily scale your solution to a large number of tenants without needing to manually reconfigure each new tenant.

In Container Apps, you manage certificates at the environment level. [Ingress](#) must also be enabled for the container app before you can bind a custom domain to it.

Request authentication and authorization

Container Apps can [validate authentication tokens on behalf of your app](#). If a request doesn't contain a token, the token isn't valid, or the request isn't authorized, you can configure Container Apps to either block the request or redirect it to your identity provider so that the user can sign in.

If your tenants use Azure Active Directory (Azure AD) as the identity provider, you can configure Container Apps to use the [/common endpoint](#) to validate user tokens. Doing so ensures that, regardless of the user's Azure AD tenant, their tokens are validated and accepted.

You can also integrate Container Apps with [Azure Active Directory B2C](#) for user authentication via partner identity providers.

For more information, see these resources:

- [Azure Container Apps authorization](#)
- [Enable authentication and authorization in Azure Container Apps with Azure Active Directory](#)

 **Note**

The authentication and authorization features in Container Apps are similar to those in Azure App Service. However, there are some differences. For more information, see [Considerations for using built-in authentication](#).

Managed identities

You can use managed identities from Azure AD to enable your container app to access other resources that are authenticated by Azure AD. When you use managed identities, your container app doesn't need to manage credentials for service-to-service communication. You can grant specific permissions to your container app's identity for role-based access control.

When you use managed identities, keep your choice of isolation model in mind. For example, suppose you share your container apps among all your tenants and deploy tenant-specific databases. You need to ensure that one tenant's application can't access a different tenant's database.

For more information, see [Managed identities in Azure Container Apps](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Daniel Larsen](#) | Principal Customer Engineer, FastTrack for Azure
- [Will Velida](#) | Customer Engineer 2, FastTrack for Azure

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Chad Kittel](#) | Principal Software Engineer, Microsoft
- [Xuhong Liu](#) | Senior Service Engineer, FastTrack for Azure

- [Aarthi Murugan](#) | Senior Program Manager, CS Tech Strategy App Innovation
- [Kendall Roden](#) | Senior Program Manager, Azure Container Apps
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Container Apps product overview](#)
- [Container Apps documentation](#)

Related resources

- [Resources for architects and developers of multitenant solutions](#)
- [Architect multitenant solutions on Azure](#)
- [Checklist for architecting and building multitenant solutions on Azure](#)

Azure Kubernetes Service (AKS) considerations for multitenancy

Azure Azure Kubernetes Service (AKS)

Azure Kubernetes Service (AKS) simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to the Azure cloud platform. As a hosted Kubernetes service, Azure handles critical tasks, like health monitoring and maintenance. The Azure platform manages the AKS control plane, and you only pay for the AKS nodes that run your applications.

AKS clusters can be shared across multiple tenants in different scenarios and ways. In some cases, diverse applications can run in the same cluster. In other cases, multiple instances of the same application can run in the same shared cluster, one for each tenant. All these types of sharing are frequently described using the umbrella term *multitenancy*. Kubernetes doesn't have a first-class concept of end-users or tenants. Still, it provides several features to help you manage different tenancy requirements.

In this article, we describe some of the features of AKS that are useful when building multitenant systems. For general guidance and best practices for Kubernetes multitenancy, see [Multi-tenancy](#) in the Kubernetes documentation.

Multitenancy types

The first step to determining how to share an AKS cluster across multiple tenants is to evaluate the patterns and tools at your disposal. In general, multitenancy in Kubernetes clusters falls into two main categories, though many variations are still possible. The Kubernetes [documentation](#) describes two common use cases for multitenancy: multiple teams and multiple customers.

Multiple teams

A common form of multitenancy is to share a cluster between multiple teams within an organization. Each team can deploy, monitor, and operate one or more solutions. These workloads frequently need to communicate with each other and with other internal or external applications that are located on the same cluster or other hosting platforms.

In addition, these workloads need to communicate with services, such as a relational database, a NoSQL repository, or a messaging system, which is hosted in the same

cluster or is running as PaaS services on Azure.

In this scenario, members of the teams often have direct access to Kubernetes resources via tools, such as [kubectl](#). Or, members have indirect access through GitOps controllers, such as [Flux](#) and [Argo CD](#), or through other types of release automation tools.

For more information on this scenario, see [Multiple teams](#) in the Kubernetes documentation.

Multiple customers

Another common form of multitenancy frequently involves a software-as-a-service (SaaS) vendor. Or, a service provider runs multiple instances of a workload for their customers, which are considered separate tenants. In this scenario, the customers don't have direct access to the AKS cluster, but they only have access to their application. Moreover, they don't even know that their application runs on Kubernetes. Cost optimization is frequently a critical concern. Service providers use Kubernetes policies, such as [resource quotas](#) and [network policies](#), to ensure that the workloads are strongly isolated from each other.

For more information on this scenario, see [Multiple customers](#) in the Kubernetes documentation.

Isolation models

According to the [Kubernetes documentation](#), a multitenant Kubernetes cluster is shared by multiple users and workloads that are commonly referred to as *tenants*. This definition includes Kubernetes clusters that different teams or divisions share within an organization. It also contains clusters that are shared by per-customer instances of a software-as-a-service (SaaS) application.

Cluster multitenancy is an alternative to managing many single-tenant dedicated clusters. The operators of a multitenant Kubernetes cluster must isolate tenants from each other. This isolation minimizes the damage that a compromised or malicious tenant can do to the cluster and to other tenants.

When several users or teams share the same cluster with a fixed number of nodes, there's a concern that one team could use more than its fair share of resources.

[Resource Quotas](#) is a tool for administrators to address this concern.

Based on the security level provided by isolation, we can distinguish between soft and hard multitenancy.

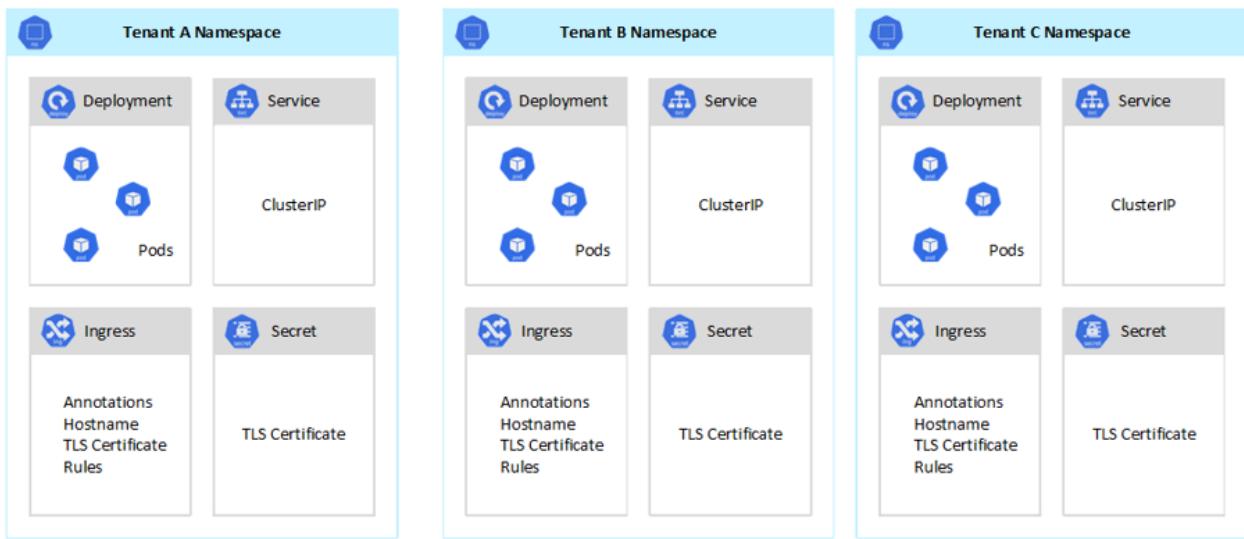
- Soft multitenancy is suitable within a single enterprise where tenants are different teams or departments that trust each other. In this scenario, isolation aims to guarantee workloads integrity, orchestrate cluster resources across different internal user groups, and defend against possible security attacks.
- Hard multi tenancy is used to describe scenarios where heterogeneous tenants don't trust each other, often from security and resource-sharing perspectives.

When you plan to build a multitenant [Azure Kubernetes Service](#) (AKS) cluster, you should consider the layers of resource isolation and multitenancy that are provided by [Kubernetes](#) :

- Cluster
- Namespace
- Node pool or node
- Pod
- Container

In addition, you should consider the security implications of sharing different resources among multiple tenants. For example, scheduling pods from different tenants on the same node could reduce the number of machines needed in the cluster. On the other hand, you might need to prevent specific workloads from being collocated. For example, you might not allow untrusted code from outside your organization to run on the same node as containers that process sensitive information.

Although Kubernetes can't guarantee perfectly secure isolation between tenants, it does offer features that may be sufficient for specific use cases. As a best practice, you should separate each tenant and its Kubernetes resources into their namespaces. You can then use [Kubernetes role-based access control](#) (RBAC) and [Network Policies](#) to enforce tenant isolation. For example, the following picture shows the typical SaaS provider model that hosts multiple instances of the same application on the same cluster, one for each tenant. Each application lives in a separate namespace.



There are several ways to design and build multitenant solutions with [Azure Kubernetes Service](#) (AKS). Each of these methods comes with its own set of tradeoffs, in terms of infrastructure deployment, network topology, and security. These methods impact the isolation level, implementation effort, operational complexity, and cost. You can apply tenant isolation in the control and data planes, based on your requirements.

Control plane isolation

If you have isolation at the control plane level, you guarantee that different tenants can't access or affect each others' resources, such as pods and services. Also, they can't impact the performance of other tenants' applications. For more information, see [Control plane isolation](#) in the Kubernetes documentation. The best way to implement isolation at the control plane level is to segregate each tenant's workload and its Kubernetes resources into a separate namespace.

According to the [Kubernetes documentation](#), a [namespace](#) is an abstraction that's used to support the isolation of groups of resources within a single cluster. Namespaces can be used to isolate tenant workloads that are sharing a Kubernetes cluster:

- Namespaces allow distinct tenant workloads to exist in their own virtual workspace, without the risk of affecting each other's work. Separate teams within an organization can use namespaces to isolate their projects from each other, because they can use the same resource names in different namespaces without the risk of name overlapping.
- [RBAC roles and role bindings](#) are namespace-scoped resources that teams can use to limit tenant users and processes to access resources and services only in their namespaces. Different teams can define roles to group lists of permissions or abilities under a single name. They then assign these roles to user accounts and service accounts, to ensure that only the authorized identities have access to the resources in a given namespace.

- [Resource quotas](#) for CPU and memory are namespaced objects. Teams can use them to ensure that workloads sharing the same cluster are strongly isolated from a system resource consumption. This method can ensure that every tenant application that runs in a separate namespace, has the resources it needs to run and avoid [noisy neighbor issues](#), which could affect other tenant applications that share the same cluster.
- [Network policies](#) are namespaced objects that teams can adopt to enforce which network traffic is allowed for a given tenant application. You can use network policies to segregate distinct workloads that share the same cluster from a networking perspective.
- Team applications that run in distinct namespaces can use different [service accounts](#), to access resources within the same cluster, external applications, or managed services.
- Use namespaces to improve performance at the control plane level. If workloads in a shared cluster are organized into multiple namespaces, the Kubernetes API has fewer items to search when running operations. This organization can reduce the latency of calls against the API server and increase the throughput of the control plane.

For more information on the isolation at the namespace level, see the following resources in the Kubernetes documentation:

- [Namespaces](#)
- [Access Controls](#)
- [Quotas](#)

Data plane isolation

Data plane isolation guarantees that pods and workloads of distinct tenants are sufficiently isolated from one another. For more information, see [Data Plane Isolation](#) in the Kubernetes documentation.

Network isolation

When you run modern, microservices-based applications in Kubernetes, you often want to control which components can communicate with each other. By default, all pods in an AKS cluster can send and receive traffic without restrictions, including other applications that share the same cluster. To improve security, you can define network rules to control the flow of traffic. Network policy is a Kubernetes specification that defines access policies for communication between Pods. You can use [network](#)

[policies](#) to segregate communications between tenant applications that share the same cluster.

Azure Kubernetes Service (AKS) provides two ways to implement network policies:

1. Azure has its implementation for network policies, called Azure network policies.
2. [Calico network policies](#) is an open-source network and network security solution founded by [Tigera](#).

Both implementations use Linux IPTables to enforce the specified policies. Network policies are translated into sets of allowed and disallowed IP pairs. These pairs are then programmed as IPTable filter rules. You can only use Azure network policies with AKS clusters that are configured with the [Azure CNI](#) network plugin. However, Calico network policies support both [Azure CNI](#) and [kubenet](#). For more information, see [Secure traffic between pods using network policies in Azure Kubernetes Service](#).

For more information, see [Network isolation](#) in the Kubernetes documentation.

Storage isolation

Azure provides a rich set of managed, platform-as-a-service (PaaS) data repositories, such as [Azure SQL Database](#) and [Azure Cosmos DB](#), and other storage services that you can use as [persistent volumes](#) for your workloads. Tenant applications running on a shared AKS cluster can [share a database or file store](#), or they can use [a dedicated data repository and storage resource](#). For more information on different strategies and approaches to manage data in a multitenant scenario, see [Architectural approaches for storage and data in multitenant solutions](#).

Workloads running on [Azure Kubernetes Service](#) (AKS) can also use persistent volumes to store data. On Azure, you can create [persistent volumes](#) as Kubernetes resources that are backed by Azure Storage. You can manually create data volumes and assign them to pods directly, or you can have AKS automatically create them using [persistent volume claims](#). AKS provides built-in storage classes to create persistent volumes that are backed by [Azure Disks](#), [Azure Files](#), and [Azure NetApp Files](#). For more information, see [Storage options for applications in Azure Kubernetes Service \(AKS\)](#). For security and resiliency reasons, you should avoid using local storage on agent nodes via [emptyDir](#) and [hostPath](#).

When AKS [built-in storage classes](#) aren't a good fit for one or more tenants, you can build custom [storage classes](#) to address different tenants requirements. These requirements include volume size, storage SKU, a service-level agreement (SLA), backup policies, and the pricing tier.

For example, you can configure a custom storage class for each tenant. You can then use it to apply tags to any persistent volume that's created in their namespace to charge back their costs to them. For more information on this scenario, see [Use Azure tags in Azure Kubernetes Service \(AKS\)](#).

For more information, see [Storage isolation](#) in the Kubernetes documentation.

Node isolation

You can configure tenant workloads to run on separate agent nodes to avoid the [Noisy Neighbor problem](#) and reduce the risk of information disclosure. In AKS, you can create a separate cluster, or just a dedicated node pool, for tenants that have strict requirements for isolation, security, regulatory compliance, and performance.

You can use [taints](#), [tolerations](#), [node labels](#), [node selectors](#), and [node affinity](#) to constrain tenants' pods to run only on a particular set of nodes or node pools.

In general, AKS provides workload isolation at various levels:

- At the kernel level, by running tenant workloads in lightweight virtual machines on shared agent nodes and using [Pod Sandboxing](#) based on [Kata Containers](#).
- At the physical level, by hosting tenant applications on dedicated clusters or node pools.
- At the hardware level, by running tenant workloads on [Azure dedicated hosts](#) that guarantee that agent node VMs run dedicated physical machines. Hardware isolation ensures that no other virtual machines are placed on the dedicated hosts, providing an additional layer of isolation for tenant workloads.

You can combine these techniques. For example, you can run per-tenant clusters and node pools in an [Azure Dedicated Host group](#) to achieve workload segregation and physical isolation at the hardware level. You can also create shared or per-tenant node pools that support [Federal Information Process Standard \(FIPS\)](#), [Confidential Virtual Machines \(CVM\)](#), or host-based encryption.

Node isolation allows you to easily associate and charge back the cost of a set of nodes or node pool to a single tenant. It's strictly related to the tenancy model that's adopted by your solution.

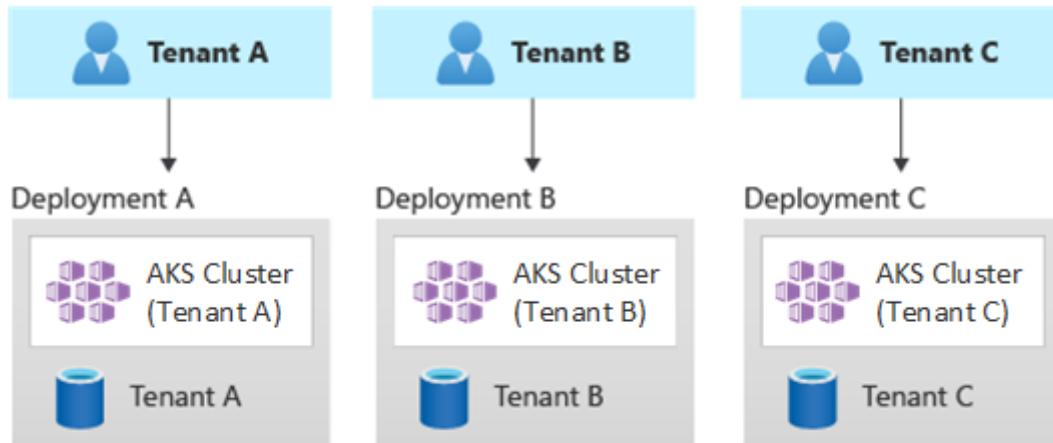
For more information, see [Node Isolation](#) in the Kubernetes documentation.

Tenancy models

Azure Kubernetes Service (AKS) provides more types of node isolation and tenancy models.

Automated single-tenant deployments

In an automated single-tenant deployment model, you deploy a dedicated set of resources for each tenant, as illustrated in this example:



Each tenant workload runs in a dedicated AKS cluster and accesses a distinct set of Azure resources. Typically, multitenant solutions that are built using this model make extensive use of [infrastructure as code](#) (IaC). For example, [Bicep](#), [Azure Resource Manager \(ARM\)](#), [Terraform](#), or the [Azure Resource Manager REST APIs](#) help initiate and coordinate the on-demand deployment of tenant-dedicated resources. You might use this approach when you need to provision an entirely separate infrastructure for each of your customers. When planning your deployment, consider using the [Deployment Stamps pattern](#).

Benefits:

- A key benefit of this approach is that the API Server of each tenant AKS cluster is separate. This approach guarantees full isolation across tenants from a security, networking, and resource consumption level. An attacker that manages to get control of a container will only have access to the containers and volumes mounted that belong to a single tenant. A full-isolation tenancy model is critical to some customers with a high regulatory compliance overhead.
- Tenants are unlikely to affect each other's system performance, which allows you to avoid the [Noisy Neighbor problem](#). This consideration includes the traffic against the API Server. The API server is a shared, critical component in any Kubernetes cluster. Custom controllers, which generate unregulated, high-volume traffic against the API server, can cause cluster instability. This instability leads to request failures, timeouts, and API retry storms. The [Uptime SLA](#) (service-level agreement) feature allows you to scale out the control plane of an AKS cluster to meet traffic

demand. Still, provisioning a dedicated cluster may be a better solution for those customers with strong requirements in terms of workload isolation.

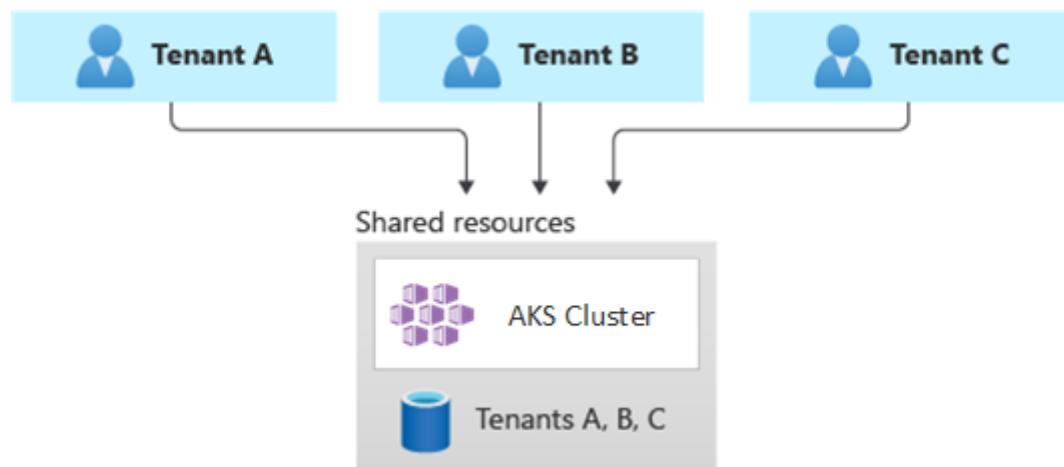
- Updates and changes can be rolled out progressively across tenants, which reduce the likelihood of a system-wide outage. Azure costs can be easily charged back to tenants, as every resource is used by a single tenant.

Risks:

- Cost efficiency is low because every tenant uses a dedicated set of resources.
- Ongoing maintenance is likely to be time-consuming, as it needs to be replicated across multiple AKS clusters, one for each tenant. Consider automating your operational processes and applying changes progressively through your environments. It would help if you also considered other cross-deployment operations, like reporting and analytics across your whole estate. Likewise, ensure you plan how to query and manipulate data across multiple deployments.

Fully multitenant deployments

In a fully multitenant deployment, a single application serves the requests of all the tenants, and all the Azure resources are shared, including the AKS cluster. In this context, you only have one set of infrastructure to deploy, monitor, and maintain. All the tenants use the resource, as illustrated in the following diagram:



Benefits:

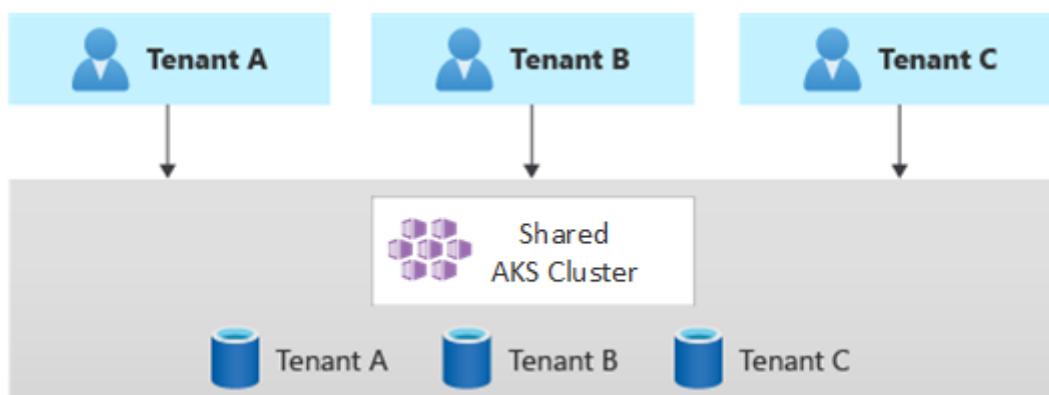
- This model is attractive because of the lower cost of operating a solution with shared components. When using this tenancy model, you may need to deploy a larger AKS cluster and adopt a higher SKU for any shared data repository to sustain the traffic generated by all tenants' resources, such as data repositories.

Risks:

- In this context, a single application handles all the tenants' requests. You should design and implement security measures to prevent tenants from flooding the application with calls. These calls could slow down the entire system and impact all the tenants.
- If the traffic profile is highly variable, you should configure the AKS cluster autoscaler to vary the number of pods and agent nodes. Base your configuration on the system resource usage, such as CPU and Memory. Alternatively, you could scale out and scale in the number of pods and cluster nodes, based on custom metrics. For example, you can explore the number of pending requests or the metrics of an external messaging system that uses [Kubernetes Event-driven Autoscaling](#) (KEDA).
- Make sure you separate the data for each tenant and implement safeguards to avoid data leakage between different tenants.
- Make sure to [track and associate Azure costs](#) to individual tenants, based on their actual usage. Third-party solutions, such as [kubecost](#), can help you calculate and break down costs across different teams and tenants.
- Maintenance can be more straightforward with a single deployment, since you only have to update one set of Azure resources and maintain a single application. However, it's also often riskier since any changes to the infrastructure or application components can affect the entire customer base.
- You should also consider scale limitations. You're more likely to hit Azure resource scale limits when you have a shared set of resources. To avoid hitting a resource quota limit, you might consider distributing your tenants across multiple Azure subscriptions.

Horizontally partitioned deployments

Alternatively, you can consider horizontally partitioning your multitenant Kubernetes application. This approach consists in sharing some solution components across all the tenants and deploying dedicated resources for individual tenants. For example, you could build a single multitenant Kubernetes application and then create individual databases, one for each tenant, as shown in this illustration:



Benefits:

- Horizontally partitioned deployments can help you mitigate the [Noisy Neighbor issue](#). Consider this approach, if you've identified that most of the traffic load on your Kubernetes application is due to specific components, which you can deploy separately, for each tenant. For example, your databases might absorb most of your system's load, because the query load is high. If a single tenant sends a large number of requests to your solution, the performance of a database might be negatively affected, but other tenants' databases (and shared components, like the application tier) remain unaffected.

Risks:

- With a horizontally partitioned deployment, you still need to consider the automated deployment and management of your components, especially the components used by a single tenant.
- This model may not provide the required level of isolation for those customers that can't afford to share resources with other tenants, for internal policies or compliance reasons.

Vertically partitioned deployments

You can take advantage of the benefits of the single-tenant and fully multitenant models by using a hybrid model that vertically partitions tenants across multiple AKS clusters or node pools. This approach provides the following advantages over the previous two tenancy models:

- You can use a combination of single-tenant and multitenant deployments. For example, you can have most of your customers share an AKS cluster and database on a multitenant infrastructure. Still, you might also deploy single-tenant infrastructures for those customers who require higher performance and isolation.
- You can deploy tenants to multiple regional AKS clusters, potentially with different configurations. This technique is most effective when you have tenants spread across different geographies.

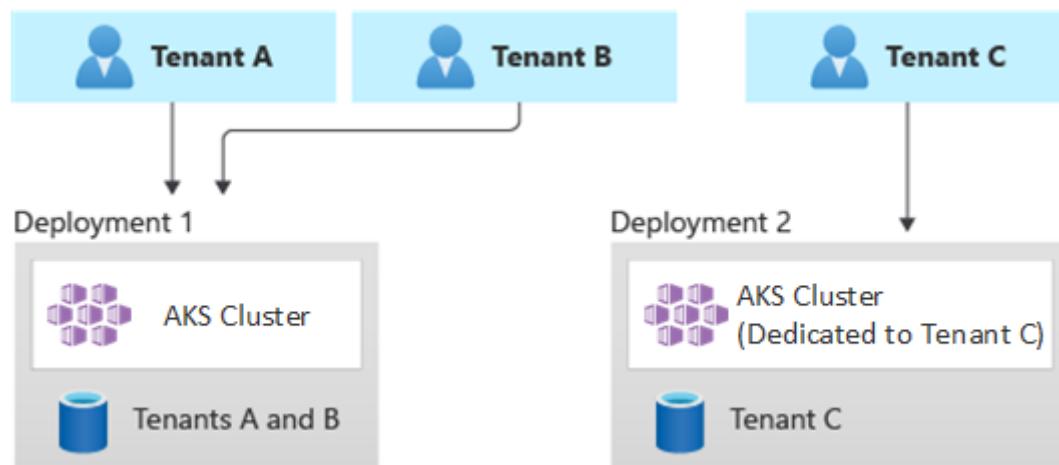
You can implement different variations of this tenancy model. For example, you can choose to offer your multitenant solution with different tiers of functionality at a different cost. Your pricing model could provide multiple SKUs, each providing an incremental level of performance and isolation, in terms of resource sharing, performance, network, and data segregation. Consider the following tiers:

- Basic tier: The tenant requests are served by a single, multitenant Kubernetes application that's shared with other tenants. Data is stored in one or more

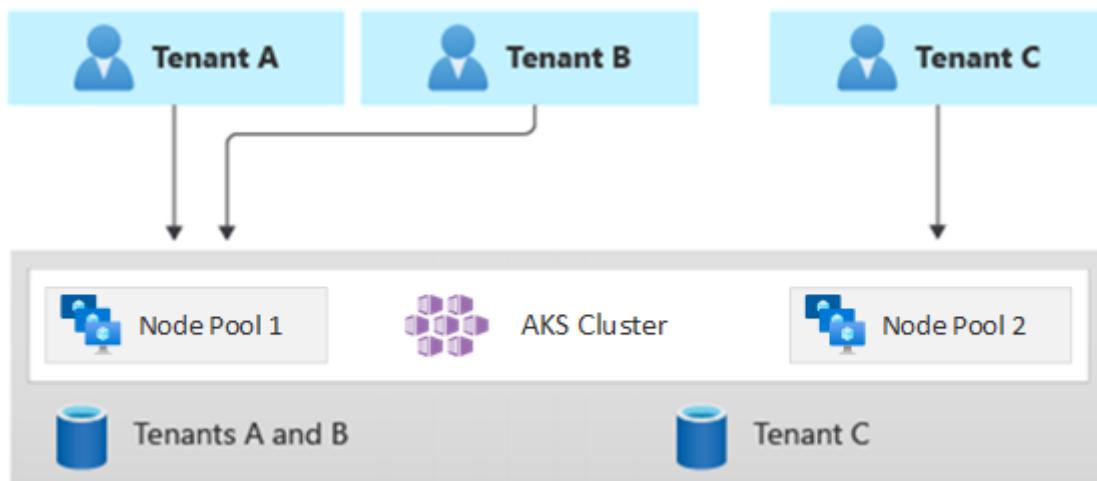
databases that are shared by all Basic-tier tenants.

- Standard tier: Tenants requests are served by a dedicated Kubernetes application that runs in a separate namespace, which provides isolation boundaries in terms of security, networking, resource consumption. All the tenants' applications, one for each tenant, share the same AKS cluster and node pool with other standard-tier customers.
- Premium tier: The tenant application runs in a dedicated node pool or AKS cluster to guarantee a higher service level agreement, better performance, and a higher degree of isolation. This tier can provide a flexible cost model based on the number and SKU of the agent nodes that are used to host the tenant application. You can use [Pod Sandboxing](#) as an alternative solution to using dedicated clusters or node pools to isolate distinct tenant workloads.

The following diagram shows a scenario where tenants A and B run on a shared AKS cluster, whereas tenant C runs on a separate AKS cluster.



Likewise, the following diagram shows a scenario where tenants A and B run on the same node pool, whereas tenant C runs on a dedicated node pool.



This model can also offer different service-level agreements for different tiers. For example, the basic tier can offer 99.9% uptime, the standard tier can offer 99.95%

uptime, and the premium tier can offer 99.99%. The higher service-level agreement (SLA) could be implemented by using services and features that enable higher availability targets.

Benefits:

- Since you're still sharing infrastructure, you can still gain some of the cost benefits of having shared multitenant deployments. You can deploy clusters and node pools that are shared across multiple basic-tier and standard-tier tenant applications, which use a cheaper VM size for agent nodes. This approach guarantees better density and cost savings. For premium-tier customers, you can deploy AKS clusters and node pools with a higher VM size and a maximum number of pod replicas and nodes at a higher price.
- You can run system services, such as [CoreDNS](#), [Konnectivity](#), or [Azure Application Gateway Ingress Controller](#), in a dedicated system-mode node pool. You can use [taints](#), [tolerations](#), [node labels](#), [node selectors](#), and [node affinity](#) to run a tenant application on one or more user-mode node pools.
- You can use [taints](#), [tolerations](#), [node labels](#), [node selectors](#), and [node affinity](#) to run shared resources. For example, you can have an ingress controller or messaging system on a dedicated node pool, with a specific VM size, autoscaler settings, and availability-zones support.

Risks:

- You need to design your Kubernetes application to support both multitenant and single-tenant deployments.
- If you plan to allow migration between infrastructures, you need to consider how you migrate customers from a multitenant deployment to their own single-tenant deployment.
- You need a consistent strategy and a single pane of glass (one viewpoint) to monitor and manage more AKS clusters.

Autoscaling

To keep up with the traffic demand that's generated by tenant applications, you can enable the [cluster autoscaler](#) to scale the agent nodes of your Azure Kubernetes Service (AKS). Autoscaling helps systems remain responsive in the following circumstances:

- The traffic load increases during specific work hours or periods of the year.
- Tenant or shared heavy loads are deployed to a cluster.
- Agent nodes become unavailable due to zonal outages.

When you enable autoscaling for a node pool, you specify a minimum and a maximum number of nodes based on the expected workload sizes. By configuring a maximum number of nodes, you can ensure enough space for all the tenant pods in the cluster, regardless of the namespace they run in.

When the traffic increases, cluster autoscaling adds new agent nodes to avoid pods going into a pending state, due to a shortage of resources in terms of CPU and memory.

Likewise, when the load diminishes, cluster autoscaling decreases the number of agent nodes in a node pool, based on the specified boundaries, which helps reduce your operational costs.

You can use pod autoscaling to scale pods automatically, based on resource demands. [Horizontal Pod Autoscaler \(HPA\)](#) automatically scales the number of pod replicas, based on CPU/memory utilization or custom metrics. With [Kubernetes Event-driven Autoscaling](#) (KEDA), you can drive the scaling of any container in Kubernetes, based on the number of events of external systems, such as [Azure Event Hubs](#) or [Azure Service Bus](#), which are used by tenant applications.

Maintenance

To reduce the risk of downtimes that may affect tenant applications during cluster or node pool upgrades, schedule AKS [Planned Maintenance](#) to occur during off-peak hours. Planned Maintenance allows you to schedule weekly maintenance windows to update the control plane of the AKS clusters that run tenant applications and node pools, which minimizing workload impact. You can schedule one or more weekly maintenance windows on your cluster by specifying a day or time range on a specific day. All maintenance operations will occur during the scheduled windows.

Security

Cluster access

When you share an AKS cluster between multiple teams within an organization, you need to implement the [principle of least privilege](#) to isolate different tenants from one another. In particular, you need to make sure that users have access only to their Kubernetes namespaces and resources when using tools, such as [kubectl](#), [Helm](#), [Flux](#), [Argo CD](#), or other types of tools.

For more information about authentication and authorization with AKS, see the following articles:

- Access and identity options for Azure Kubernetes Service (AKS)
- AKS-managed Microsoft Entra integration
- Control access to cluster resources using Kubernetes role-based access control and Microsoft Entra identities in Azure Kubernetes Service

Workload identity

If you host multiple tenant applications on a single AKS cluster, and each is in a separate namespace, then each workload should use a different [Kubernetes service account](#) and credentials to access the downstream Azure services. *Service accounts* are one of the primary user types in Kubernetes. The Kubernetes API holds and manages service accounts. Service account credentials are stored as Kubernetes secrets, which allows them to be used by authorized pods to communicate with the API Server. Most API requests provide an authentication token for a service account or a normal user account.

Tenant workloads deployed to AKS clusters can use Microsoft Entra application credentials to access Microsoft Entra ID-protected resources, such as Azure Key Vault and Microsoft Graph. [Microsoft Entra Workload ID for Kubernetes](#) integrates with the Kubernetes native capabilities to federate with any external identity providers.

Pod Sandboxing

AKS includes a mechanism called [Pod Sandboxing](#) that provides an isolation boundary between the container application and the shared kernel and compute resources of the container host, like CPU, memory, and networking. Pod Sandboxing complements other security measures or data protection controls to help tenant workloads secure sensitive information and meet regulatory, industry, or governance compliance requirements, like Payment Card Industry Data Security Standard (PCI DSS), International Organization for Standardization (ISO) 27001, and Health Insurance Portability and Accountability Act (HIPAA).

Deploying applications on separate clusters or node pools enables you to strongly isolate the tenant workloads of different teams or customers. Using multiple clusters and node pools might be suitable for the isolation requirements of many organizations and SaaS solutions, but there are scenarios in which a single cluster with shared VM node pools is more efficient, for example, when you're running untrusted and trusted pods on the same node or co-locating DaemonSets and privileged containers on the same node for faster local communication and functional grouping. [Pod Sandboxing](#) can help you to strongly isolate tenant applications on the same cluster nodes without needing to run these workloads in separate clusters or node pools. Other methods require that you recompile your code or cause other compatibility problems, but Pod

Sandboxing in AKS can run any container unmodified inside an enhanced security VM boundary.

Pod Sandboxing on AKS is based on [Kata Containers](#) that run on the [Azure Linux container host for AKS](#) stack to provide hardware-enforced isolation. Kata Containers on AKS are built on a security-hardened Azure hypervisor. The isolation per pod is achieved via a nested lightweight Kata VM that utilizes resources from a parent VM node. In this model, each Kata pod gets its own kernel in a nested Kata guest VM. This model enables you to place many Kata containers in a single guest VM while continuing to run containers in the parent VM. The model provides a strong isolation boundary in a shared AKS cluster.

For more information, see:

- [Pod Sandboxing with Azure Kubernetes Service \(AKS\)](#)
- [Support for Kata VM Isolated Containers on AKS for Pod Sandboxing](#)

Azure Dedicated Host

[Azure Dedicated Host](#) is a service that provides physical servers that are dedicated to a single Azure subscription and provide hardware isolation at the physical-server level. These dedicated hosts can be provisioned within a region, availability zone, and fault domain, and you can place VMs directly into the provisioned hosts.

You can achieve several benefits by using Azure Dedicated Host with AKS, including the following:

- Hardware isolation ensures that no other VMs are placed on the dedicated hosts, which provides an additional layer of isolation for tenant workloads. Dedicated hosts are deployed in the same datacenters and share the same network and underlying storage infrastructure as other non-isolated hosts.
- Azure Dedicated Host provides control over maintenance events that are initiated by the Azure platform. You can choose a maintenance window to reduce the impact on services and help ensure the availability and privacy of tenant workloads.

Azure Dedicated Host can help SaaS providers ensure tenant applications meet regulatory, industry, and governance compliance requirements for securing sensitive information. For more information, see [Add Azure Dedicated Host to an Azure Kubernetes Service \(AKS\) cluster](#).

Confidential Virtual Machines

You can use [Confidential Virtual Machines \(CVMs\)](#) to add one or more node pools to your AKS cluster to address tenants' strict isolation, privacy, and security requirements. [CVMs](#) use a hardware-based [trusted execution environment \(TEE\)](#). [AMD Secure Encrypted Virtualization - Secure Nested Paging \(SEV-SNP\)](#) confidential VMs deny the hypervisor and other host management code access to VM memory and state, adding a layer of defense in-depth protection against operator access. For more information, see [Use CVMs in an AKS cluster](#).

Federal Information Process Standards (FIPS)

[FIPS 140-3](#) is a US government standard that defines minimum security requirements for cryptographic modules in information technology products and systems. By enabling [FIPS compliance for AKS node pools](#), you can enhance the isolation, privacy, and security of your tenant workloads. FIPS compliance ensures the use of validated cryptographic modules for encryption, hashing, and other security-related operations. With FIPS-enabled AKS node pools, you can meet regulatory and industry compliance requirements by employing robust cryptographic algorithms and mechanisms. Azure provides documentation on how to enable FIPS for AKS node pools, enabling you to strengthen the security posture of your multitenant AKS environments. For more information, see [Enable FIPS for AKS node pools](#).

Bring your own keys (BYOK) with Azure disks

Azure Storage encrypts all data in a storage account at rest, including the OS and data disks of an AKS cluster. By default, data is encrypted with Microsoft-managed keys. For more control over encryption keys, you can supply customer-managed keys to use for encryption at rest of the OS and data disks of your AKS clusters. For more information, see:

- [BYOK with Azure disks in AKS](#)
- [Server-side encryption of Azure Disk Storage](#)

Host-based encryption

[Host-based encryption](#) on AKS further strengthens tenant workload isolation, privacy, and security. When you enable host-based encryption, AKS encrypts data at rest on the underlying host machines, helping to ensure that sensitive tenant information is protected from unauthorized access. Temporary disks and ephemeral OS disks are encrypted at rest with platform-managed keys when you enable end-to-end encryption.

In AKS, OS and data disks use server-side encryption with platform-managed keys by default. The caches for these disks are encrypted at rest with platform-managed keys. You can specify your own [key encryption key \(KEK\)](#) to encrypt the [data protection key \(DEK\)](#) by using envelope encryption, also known as *wrapping*. The cache for the OS and data disks are also encrypted via the [BYOK](#) that you specify.

Host-based encryption adds a layer of security for multitenant environments. Each tenant's data in the OS and data disk caches is encrypted at rest with either customer-managed or platform-managed keys, depending on the selected disk encryption type. For more information, see:

- [Host-based encryption on AKS](#)
- [BYOK with Azure disks in AKS](#)
- [Server-side encryption of Azure Disk Storage](#)

Networking

Restrict network access to the API server

In Kubernetes, the API server receives requests to perform actions in the cluster, such as creating resources or scaling the number of nodes. When sharing an AKS cluster across multiple teams within an organization, protect access to the control plane by using one of the following solutions.

Private AKS clusters

By using a private AKS cluster, you can make sure the network traffic between your API server and your node pools remains within your virtual network. In a private AKS cluster, the control plane or API server has an internal IP address that's only accessible via an [Azure private endpoint](#), which is located in the same virtual network of the AKS cluster. Likewise, any virtual machine in the same virtual network can privately communicate with the control plane via the private endpoint. For more information, see [Create a private Azure Kubernetes Service cluster](#).

Authorized IPs

The second option to improve cluster security and minimize attacks is using [authorized IPs](#). This approach restricts the access to the control plane of a public AKS cluster to a well-known list of Internet Protocol (IP) addresses and Classless Inter-Domain Routing (CIDR). When you use authorized IPs, they're still publicly exposed, but access is limited

to a set of IP ranges. For more information, see [Secure access to the API server using authorized IP address ranges in Azure Kubernetes Service \(AKS\)](#).

Private Link integration

Azure Private Link Service (PLS) is an infrastructure component that allows applications to privately connect to a service via an [Azure private endpoint \(PE\)](#) that's defined in a virtual network and connected to the frontend IP configuration of an [Azure Load Balancer \(ALB\)](#) instance. With [Azure Private Link](#), service providers can securely provide their services to their tenants that can connect from within Azure or on-premises, without data exfiltration risks.

You can use [Azure Private Link Service Integration](#) to provide tenants with private connectivity to their AKS-hosted workloads in a secure way, without the need to expose any public endpoint on the public internet.

For general guidance on how you can configure Private Link for an Azure-hosted multitenant solution, see [Multitenancy and Azure Private Link](#).

Reverse proxies

A [reverse proxy](#) is a load balancer and an [API gateway](#) that is typically used in front of tenant applications to secure, filter, and dispatch incoming requests. Popular reverse proxies support features such as load balancing, SSL termination, and layer 7 routing. Reverse proxies are typically implemented to help increase security, performance, and reliability. Popular reverse proxies for Kubernetes include the following implementations:

- [NGINX Ingress Controller](#) is a popular reverse proxy server that supports advanced features, such as load balancing, SSL termination, and layer 7 routing.
- [Traefik Kubernetes Ingress provider](#) is a Kubernetes Ingress controller that can be used to manage access to cluster services by supporting the ingress specification.
- [HAProxy Kubernetes Ingress Controller](#) is yet another reverse proxy for Kubernetes, which supports standard features such as TLS termination, URL-path-based routing, and more.
- [Azure Application Gateway Ingress Controller \(AGIC\)](#) is a [Kubernetes](#) application, which makes it possible for Azure Kubernetes Service (AKS) customers to leverage Azure's native [Application Gateway](#) L7 load-balancer, to expose tenant applications to the public internet or internally to other systems that run in a virtual network.

When using an AKS-hosted reverse proxy to secure and handle incoming requests to multiple tenant applications, consider the following recommendations:

- Host the reverse proxy on a dedicated node pool that's configured to use a VM size with a high-network bandwidth and [accelerated networking](#) enabled.
- Configure the node pool that's hosting your reverse proxy for autoscaling.
- To avoid increased latency and timeouts for tenant applications, define an autoscaling policy so that the number of ingress controller pods can instantly expand and contract to match traffic fluctuations.
- Consider sharding the incoming traffic to tenant applications, across multiple instances of your ingress controller, to increase the scalability and segregation level.

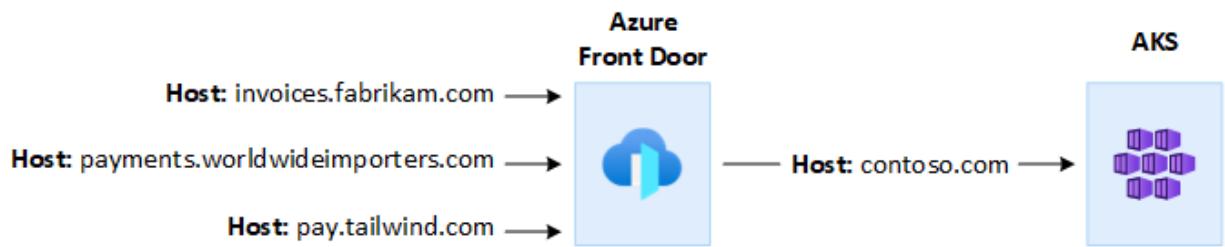
When using the [Azure Application Gateway Ingress Controller \(AGIC\)](#), consider implementing the following best practices:

- Configure the [Application Gateway](#) that's used by the ingress controller for [Autoscaling](#). With autoscaling enabled, the Application Gateway and WAF v2 SKUs scale out or in, based on application traffic requirements. This mode offers better elasticity to your application and eliminates the need to guess the application gateway size or instance count. This mode also allows you to save cost, by not requiring the gateway to run at peak-provisioned capacity for the expected maximum traffic load. You must specify a minimum (and optionally maximum) instance count.
- Consider deploying multiple instances of the [Application Gateway Ingress Controller \(AGIC\)](#), each associated to a separate [Application Gateway](#), when the number of tenant applications exceeds the [maximum amount of sites](#). Assuming that each tenant application runs in a dedicated namespace, use [multiple namespace support](#) to spread tenant applications across more instances of the [Application Gateway Ingress Controller \(AGIC\)](#).

Integration with Azure Front Door

[Azure Front Door](#) is a global layer-7 load balancer and Microsoft's modern cloud content delivery network (CDN) that provides fast, reliable, and secure access between users and web applications across the globe. Azure Front Door supports features such as request acceleration, SSL termination, response caching, WAF at the edge, URL-based routing, rewrite, and redirections that you can exploit when exposing AKS-hosted multitenant applications to the public internet.

For example, you might want to use an AKS-hosted multitenant application to serve all the customers' requests. In this context, you can use Azure Front Door to manage multiple custom domains, one for each tenant. You can terminate SSL connections on the edge and route all the traffic to the AKS-hosted multitenant application that's configured with a single hostname.



You can configure Azure Front Door to modify the [request origin host header](#) to match the domain name of the backend application. The original `Host` header sent by the client is propagated through the `x-Forwarded-Host` header, and the code of the multitenant application can use this information to [map the incoming request to the correct tenant](#).

[Azure Web Application Firewall \(WAF\)](#), on Azure Front Door, provides centralized protection for web applications. You can use Azure WAF to defend AKS-hosted tenant applications that expose a public endpoint on the internet from malicious attacks.

You can configure Azure Front Door Premium to privately connect to one or more tenant applications that run on an AKS cluster, via an internal load balancer origin, by using [Azure Private Link Service](#). For more information, see [Connect Azure Front Door Premium to an internal load balancer origin with Private Link](#).

Outbound connections

When AKS-hosted applications connect to a large number of databases or external services, the cluster might be at risk of SNAT port exhaustion. [SNAT Ports](#) generate unique identifiers that are used to maintain distinct flows that are initiated by applications that run on the same set of compute resources. By running several tenant applications on a shared Azure Kubernetes Service cluster, you might make a high number of outbound calls, which can lead to a SNAT port exhaustion. An AKS cluster can handle outbound connections in three different ways:

- [Azure Public Load Balancer](#): By default, AKS provisions a Standard SKU Load Balancer to be set up and used for egress connections. However, the default setup may not meet the requirements of all scenarios, if public IPs are disallowed or if additional hops are required for egress. By default, the public Load Balancer gets created with a default public IP address that is used by the [outbound rules](#). Outbound rules allow you to explicitly define source network address translation (SNAT) for a public standard load balancer. This configuration allows you to use the public IP(s) of your load balancer to provide outbound internet connectivity for your backend instances. When necessary, to avoid [SNAT port exhaustion](#), you can configure the outbound rules of the public load balancer to use additional public

IP addresses. For more information, see [Use the frontend IP address of a load balancer for outbound via outbound rules](#).

- [Azure NAT Gateway](#): You can configure an AKS cluster to use Azure NAT Gateway to route egress traffic from tenant applications. NAT Gateway allows up to 64,512 outbound UDP and TCP traffic flows per public IP address, with a maximum of 16 IP addresses. To avoid the risk of SNAT port exhaustion when using a NAT Gateway to handle outbound connections from an AKS cluster, you can associate more public IP addresses or a [public IP address prefix](#) to the gateway. For more information, see [Azure NAT Gateway considerations for multitenancy](#).
- [User-defined route \(UDR\)](#): You can customize an AKS cluster's egress route to support custom network scenarios, such as those that disallow public IPs and require the cluster to sit behind a network virtual appliance (NVA). When you configure a cluster for [user-defined routing](#), AKS won't automatically configure egress paths. The egress setup must be done by you. For example, you'd route egress traffic through an [Azure Firewall](#). The AKS cluster must be deployed into an existing virtual network, with a subnet that has been previously configured. When you aren't using a standard load balancer (SLB) architecture, you must establish explicit egress. As such, this architecture requires explicitly sending egress traffic to an appliance, like a firewall, gateway, or proxy. Or, the architecture allows the network address translation (NAT) to be done by a public IP that's assigned to the standard load balancer or appliance.

Monitoring

You can use [Azure Monitor](#) and [Container Insights](#) to monitor tenant applications that run on a shared AKS cluster, and to calculate cost breakdowns on individual namespaces. Azure Monitor allows you to monitor the health and performance of Azure Kubernetes Service (AKS). It includes the collection of [logs and metrics](#), telemetry analysis, and visualizations of collected data, to identify trends and to configure alerting to proactively notify you of critical issues. You can enable [Container insights](#) to expand on this monitoring.

You can also adopt open-source tools, such as [Prometheus](#) and [Grafana](#), which are widely used by the community for Kubernetes monitoring. Or, you can adopt other third-party tools for monitoring and observability.

Costs

Cost governance is the continuous process of implementing policies to control costs. In the Kubernetes context, there are several ways organizations can control and optimize

costs. These include native Kubernetes tooling to manage and govern resource usage and consumption and to proactively monitor and optimize the underlying infrastructure. When calculating per-tenant costs, you should consider the costs associated with any resource that's used by a tenant application. The approach you follow to charge fees back to the tenants depends on the tenancy model adopted by your solution. Further details are explained with the following tenancy models:

- Fully multitenant: When a single, multitenant application serves all the tenant requests, it's your responsibility to keep track of resource consumption and the requests number generated by each tenant. You then charge your customers accordingly.
- Dedicated cluster: When a cluster is dedicated to a single tenant, it's easy to charge costs of Azure resources back to the customer. The total cost of ownership depends on many factors, which include the number and size of virtual machines, the networking costs due to network traffic, public IP addresses, load balancers, the storage services, such as managed disks or Azure files used by the tenant solution, and so on. You can tag an AKS cluster and its resources in the node resource group to facilitate cost charging operations. For more information, see [Add tags to the cluster](#).
- Dedicated node pool: You can apply an Azure tag to a new or existing node pool that's dedicated to a single tenant. Tags applied to a node pool are applied to each node within the node pool and are persisted through upgrades. Tags are also applied to new nodes that are added to a node pool during scale-out operations. Adding a tag can help with tasks like policy tracking or cost charging. For more information, see [Adding tags to node pools](#).
- Other resources: you can use tags to associate costs of dedicated resources to a given tenant. In particular, you can tag Public IPs, files, and disks using a Kubernetes manifest. Tags set in this way will maintain the Kubernetes values, even if you update them later by using another method. When public IPs, files, or disks are removed through Kubernetes, any tags that are set by Kubernetes are removed. Tags on those resources that aren't tracked by Kubernetes remain unaffected. For more information, see [Add tags by using Kubernetes](#).

You can use open-source tools, such as [KubeCost](#), to monitor and govern an AKS cluster cost. Cost allocation can be scoped to a deployment, service, label, pod, and namespace, which will give you flexibility in how you chargeback or showback users of the cluster. For more information, see [Cost governance with Kubecost](#).

For more information on the measurement, allocation, and optimization of costs for a multitenant application, see [Architectural approaches for cost management and allocation in a multitenant solution](#). For general guidance on cost optimization, see the Azure Well-Architected Framework article, [Overview of the cost optimization pillar](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Ed Price](#) | Senior Content Program Manager
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure

Next steps

Review [Resources for architects and developers of multitenant solutions](#).

Use Azure Front Door in a multitenant solution

Article • 04/05/2023

Azure Front Door is a modern cloud content delivery network (CDN) that provides fast, reliable access between users and applications' static and dynamic web content across the globe. This article describes some of the features of Azure Front Door that are useful when you work in multitenant systems. It also provides links to additional guidance and examples.

When you use Azure Front Door as part of a multitenant solution, you need to make some decisions based on your solution's design and requirements. You need to consider the following factors:

- How many tenants do you have, and how many do you expect to have in the future?
- Do you share your application tier among multiple tenants, deploy many single-tenant application instances, or deploy separate deployment stamps that are shared by multiple tenants?
- Do your tenants want to bring their own domain names?
- Will you use wildcard domains?
- Do you need to use your own TLS certificates, or will Microsoft manage your TLS certificates?
- Have you considered the [quotas and limits](#) that apply to Azure Front Door? Do you know which limits you'll approach as you grow? If you suspect that you'll approach these limits, consider using multiple Azure Front Door profiles. Or consider whether you can change the way that you use Azure Front Door to avoid the limits. Note that the Premium SKU has higher limits than the Standard SKU.
- Do you or your tenants have requirements for IP address filtering, geo-blocking, or customizing WAF rules?
- Are all your tenants' application servers internet-facing?

Features of Azure Front Door that support multitenancy

This section describes several key features of Azure Front Door that are useful for multitenant solutions. It describes how Azure Front Door can help you configure custom domains, including information about wildcard domains and TLS certificates. It also

summarizes how you can use Azure Front Door routing capabilities to support multitenancy.

Custom domains

Azure Front Door provides a default host name for each endpoint that you create, for example, `contoso.z01.azurefd.net`. For most solutions, you instead associate your own domain name with the Azure Front Door endpoint. Custom domain names enable you to use your own branding and configure routing based on the host name that's provided in a client's request.

In a multitenant solution, you might use tenant-specific domain names or subdomains, and configure Azure Front Door to route the tenant's traffic to the correct origin group for that tenant's workload. For example, you might configure a custom domain name like `tenant1.app.contoso.com`. With Azure Front Door, you can configure multiple custom domains on a single Azure Front Door profile.

For more information, see [Add a custom domain to your Front Door](#).

Wildcard domains

Wildcard domains simplify the configuration of DNS records and Azure Front Door traffic routing configuration when you use a shared stem domain and tenant-specific subdomains. For example, suppose your tenants access their applications by using subdomains like `tenant1.app.contoso.com` and `tenant2.app.contoso.com`. You can configure a wildcard domain, `*.app.contoso.com`, instead of configuring each tenant-specific domain individually.

Azure Front Door supports creating custom domains that use wildcards. You can then configure a route for requests that arrive on the wildcard domain. When you onboard a new tenant, you don't need to reconfigure your DNS servers, issue new TLS certificates, or update your Azure Front Door profile's configuration.

Wildcard domains work well if you send all your traffic to a single origin group. But if you have separate stamps of your solution, a single-level wildcard domain isn't sufficient. You either need to use multi-level stem domains or supply extra configuration by, for example, overriding the routes to use for each tenant's subdomain. For more information, see [Considerations when using domain names in a multitenant solution](#).

Azure Front Door doesn't issue managed TLS certificates for wildcard domains, so you need to purchase and supply your own certificate.

For more information, see [Wildcard domains](#).

Managed TLS certificates

Acquiring and installing TLS certificates can be complex and error prone. And TLS certificates expire after a period of time, usually one year, and need to be reissued and reinstalled to avoid disruption to application traffic. When you use Azure Front Door managed TLS certificates, Microsoft is responsible for issuing, installing, and renewing certificates for your custom domain.

Your origin application might be hosted on another Azure service that also provides managed TLS certificates, like Azure App Service. Azure Front Door transparently works with the other service to synchronize your TLS certificates.

If you allow your tenants to provide their own custom domains and you want Azure Front Door to issue certificates for these domain names, your tenants shouldn't configure CAA records on their DNS servers that might block Azure Front Door from issuing certificates on their behalf. For more information, see [TLS/SSL certificates](#).

For more information about TLS certificates, see [End-to-end TLS with Azure Front Door](#).

Routing

A multitenant application might have one or more application stamps that serve the tenants. Stamps are frequently used to enable multi-region deployments and to support scaling a solution to a large number of tenants.

Azure Front Door has a powerful set of routing capabilities that can support a number of multitenant architectures. You can use routing to distribute traffic among origins within a stamp, or to send traffic to the correct stamp for a specific tenant. You can configure routing based on individual domain names, wildcard domain names, and URL paths. And you can use the rules engine to further customize routing behavior.

For more information, see [Routing architecture overview](#).

Rules engine

You can use the Azure Front Door rules engine to customize how Azure Front Door processes requests at the network edge. The rules engine enables you to run small blocks of logic within the Azure Front Door request-processing pipeline. You can use the rules engine to override the routing configuration for a request. You can also use the

rules engine to modify elements of the request before it's sent to the origin, and to modify some parts of the response before it's returned to the client.

For example, suppose you deploy a multitenant application tier in which you also use tenant-specific wildcard subdomains, as described in the following example scenarios. You might use the rules engine to extract the tenant identifier from the request subdomain and add it to a request header. This rule could help the application tier determine which tenant the request came from.

For more information, see [What is the Azure Front Door rules engine?](#).

Example scenarios

The following example scenarios illustrate how you can configure various multitenant architectures in Azure Front Door, and how the decisions you make can affect your DNS and TLS configuration.

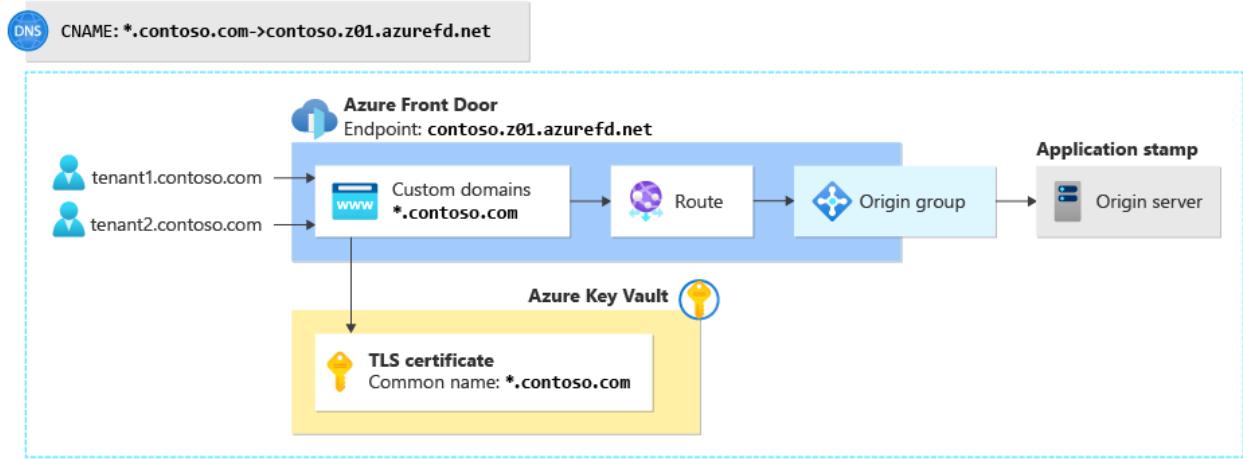
Many multitenant solutions implement the [Deployment Stamps pattern](#). When you use this deployment approach, you typically deploy a single shared Azure Front Door profile and use Azure Front Door to route incoming traffic to the appropriate stamp. This deployment model is the most common one, and scenarios 1 through 4 in this article show how you can use it to meet a range of requirements.

In some situations, however, you might deploy an Azure Front Door profile in each stamp of your solution. [Scenario 5](#) describes this deployment model.

Scenario 1: Provider-managed wildcard domain, single stamp

Contoso is building a small multitenant solution. The company deploys a single stamp in a single region, and that stamp serves all of its tenants. All requests are routed to the same application server. Contoso decided to use wildcard domains for all tenants, like `tenant1.contoso.com` and `tenant2.contoso.com`.

Contoso deploys Azure Front Door by using this configuration:



DNS configuration

One-time configuration: Contoso configures two DNS entries:

- A wildcard TXT record for *.contoso.com. It's set to the value that's specified by Azure Front Door during the custom domain onboarding process.
- A wildcard CNAME record, *.contoso.com, that's an alias for Contoso's Azure Front Door endpoint: contoso.z01.azurefd.net.

When a new tenant is onboarded: No additional configuration is required.

TLS configuration

One-time configuration: Contoso buys a wildcard TLS certificate, adds it to a key vault, and grants Azure Front Door access to the vault.

When a new tenant is onboarded: No additional configuration is required.

Azure Front Door configuration

One-time configuration: Contoso creates an Azure Front Door profile and a single endpoint. They add one custom domain with the name *.contoso.com and associate their wildcard TLS certificate with the custom domain resource. They then configure a single origin group that contains a single origin for their application server. Finally, they configure a route to connect the custom domain to the origin group.

When a new tenant is onboarded: No additional configuration is required.

Benefits

- This configuration is relatively easy to configure and provides customers with Contoso-branded URLs.
- The approach supports a large number of tenants.
- When a new tenant is onboarded, Contoso doesn't need to make changes to Azure Front Door, DNS, or TLS certificates.

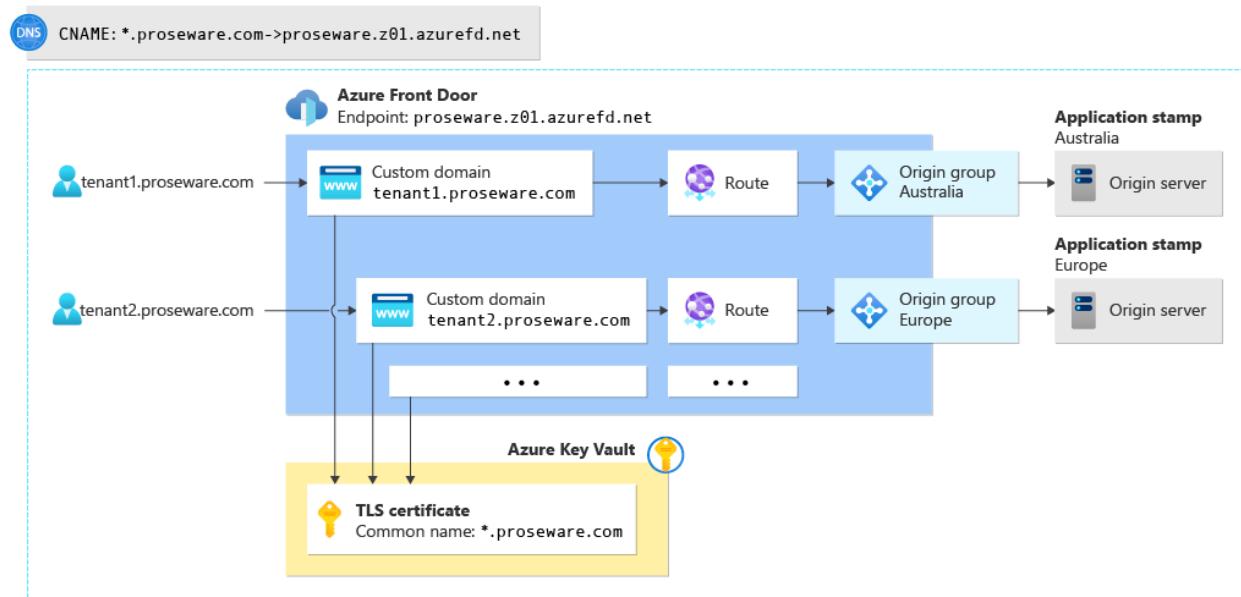
Drawbacks

- This approach doesn't easily scale beyond a single application stamp or region.
- Contoso needs to buy a wildcard TLS certificate and renew and install the certificate when it expires.

Scenario 2: Provider-managed wildcard domain, multiple stamps

Proseware is building a multitenant solution across multiple stamps that are deployed into both Australia and Europe. All requests within a single region are served by the stamp in that region. Like Contoso, Proseware decided to use wildcard domains for all tenants, like `tenant1.proseware.com` and `tenant2.proseware.com`.

Proseware deploys Azure Front Door by using this configuration:



DNS configuration

One-time configuration: Proseware configures two DNS entries:

- A wildcard TXT record for *.proseware.com. It's set to the value that's specified by Azure Front Door during the custom domain onboarding process.

- A wildcard CNAME record, *.proseware.com, that's an alias for Proseware's Azure Front Door endpoint: proseware.z01.azurefd.net.

When a new tenant is onboarded: No additional configuration is required.

TLS configuration

One-time configuration: Proseware buys a wildcard TLS certificate, adds it to a key vault, and grants Azure Front Door access to the vault.

When a new tenant is onboarded: No additional configuration is required.

Azure Front Door configuration

One-time configuration: Proseware creates an Azure Front Door profile and a single endpoint. The company configures multiple origin groups, one per application stamp/server in each region.

When a new tenant is onboarded: Proseware adds a custom domain resource to Azure Front Door. They use the name *.proseware.com and associate their wildcard TLS certificate with the custom domain resource. They then create a route to specify which origin group (stamp) that tenant's requests should be directed to. In the preceding diagram, tenant1.proseware.com routes to the origin group in the Australia region, and tenant2.proseware.com routes to the origin group in the Europe region.

Benefits

- When new tenants are onboarded, no DNS or TLS configuration changes are required.
- Proseware maintains a single instance of Azure Front Door to route traffic to multiple stamps across multiple regions.

Drawbacks

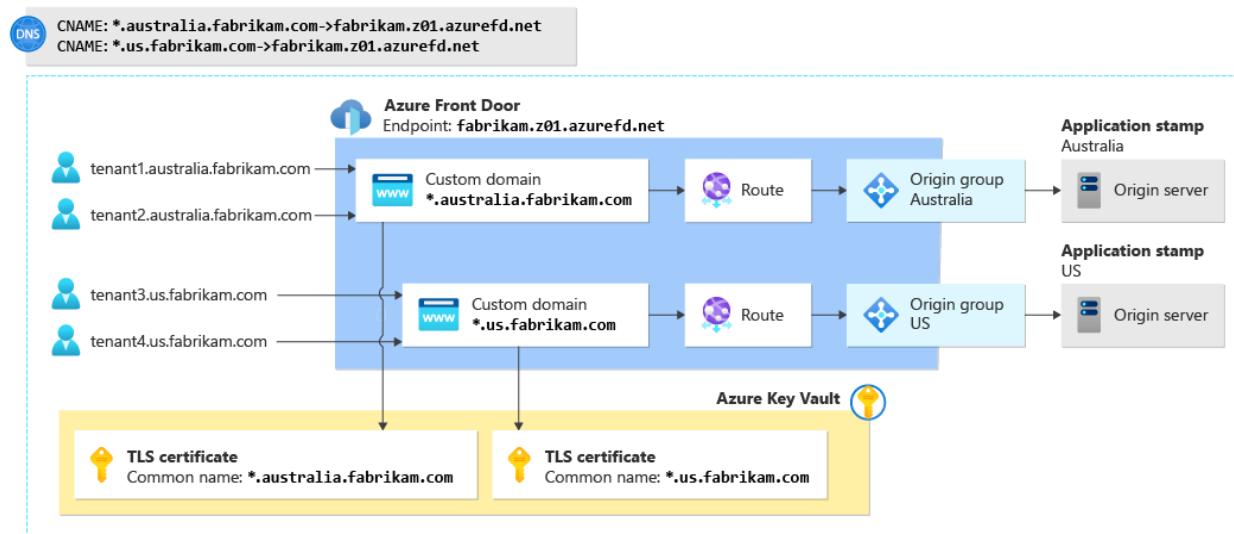
- Proseware needs to reconfigure Azure Front Door every time a new tenant is onboarded.
- Proseware needs to pay attention to [Azure Front Door quotas and limits](#), especially on the number of routes and custom domains, and the [composite routing limit](#).
- Proseware needs to buy a wildcard TLS certificate.
- Proseware is responsible for renewing and installing the certificate when it expires.

Scenario 3: Provider-managed, stamp-based wildcard subdomains

Fabrikam is building a multitenant solution. The company deploys stamps in Australia and the United States. All requests within a single region will be served by the stamp in that region. Fabrikam will use stamp-based stem domains, like

`tenant1.australia.fabrikam.com`, `tenant2.australia.fabrikam.com`, and
`tenant3.us.fabrikam.com`.

The company deploys Azure Front Door by using this configuration:



DNS configuration

One-time configuration: Fabrikam configures the following two wildcard DNS entries for each stamp.

- A wildcard TXT record for each stamp: `*.australia.fabrikam.com` and `*.us.fabrikam.com`. They're set to the values specified by Azure Front Door during the custom domain onboarding process.
- A wildcard CNAME record for each stamp, `*.australia.fabrikam.com` and `*.us.fabrikam.com`, which are both aliases for the Azure Front Door endpoint: `fabrikam.z01.azurefd.net`.

When a new tenant is onboarded: No additional configuration is required.

TLS configuration

One-time configuration: Fabrikam buys a wildcard TLS certificate for each stamp, adds them to a key vault, and grants Azure Front Door access to the vault.

When a new tenant is onboarded: No additional configuration is required.

Azure Front Door configuration

One-time configuration: Fabrikam creates an Azure Front Door profile and a single endpoint. They configure an origin group for each stamp. They create custom domains, using a wildcard, for each stamp-based subdomain: *.australia.fabrikam.com and *.us.fabrikam.com. They create a route for each stamp's custom domain to send traffic to the appropriate origin group.

When a new tenant is onboarded: No additional configuration is required.

Benefits

- This approach enables Fabrikam to scale to large numbers of tenants across multiple stamps.
- When new tenants are onboarded, no DNS or TLS configuration changes are required.
- Fabrikam maintains a single instance of Azure Front Door to route traffic to multiple stamps across multiple regions.

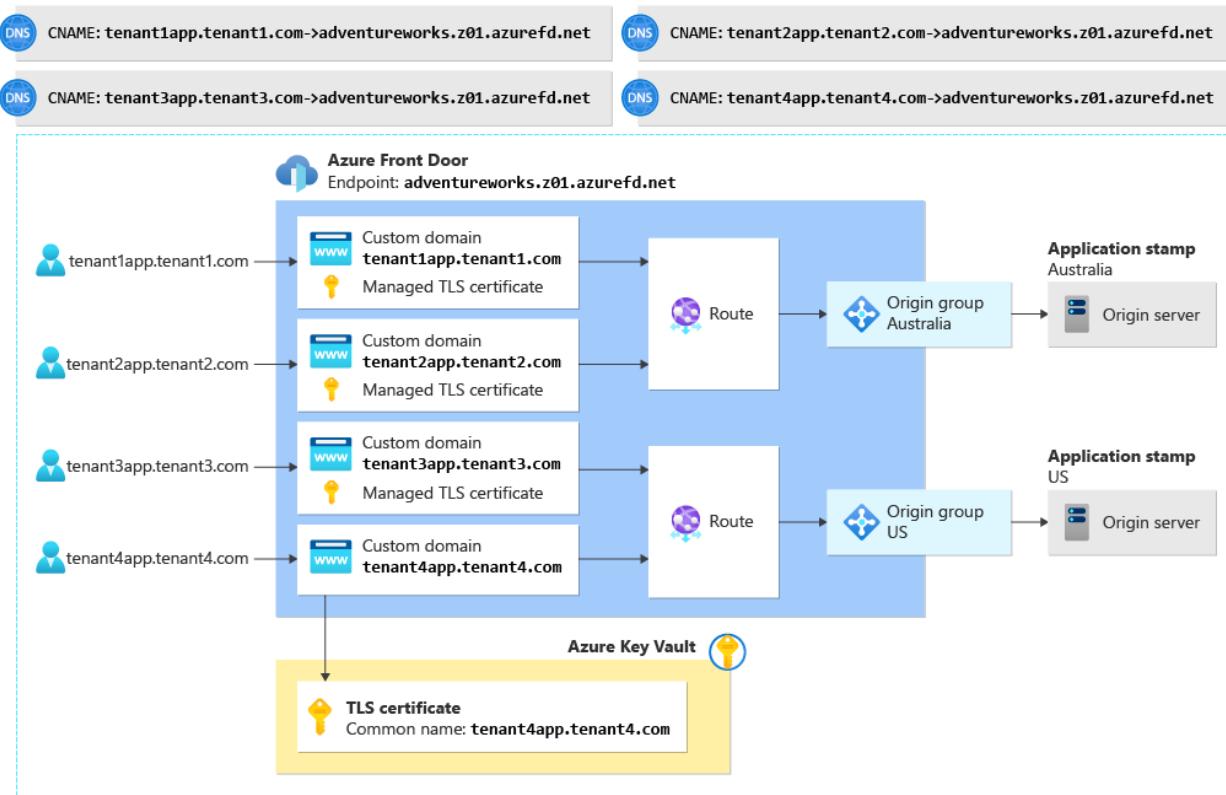
Drawbacks

- Because URLs use a multipart stem domain structure, URLs can be more complex for users to work with.
- Fabrikam needs to buy multiple wildcard TLS certificates.
- Fabrikam is responsible for renewing and installing the TLS certificates when they expire.

Scenario 4: Vanity domains

Adventure Works Cycles is building a multitenant solution. The company deploys stamps in multiple regions, like Australia and the United States. All requests within a single region will be served by the stamp in that region. Adventure Works will allow its tenants to bring their own domain names. For example, tenant 1 might configure a custom domain name like tenant1app.tenant1.com.

The company deploys Azure Front Door by using this configuration:



DNS configuration

One-time configuration: None.

When a new tenant is onboarded: The tenant needs to create two records on its own DNS server:

- A TXT record for domain validation. For example, tenant 1 needs to configure a TXT record named `tenant1app.tenant1.com` and set it to the value specified by Azure Front Door during the custom domain onboarding process.
- A CNAME record that's aliased to the Adventure Works Azure Front Door endpoint. For example, tenant 1 needs to configure a CNAME record named `tenant1app.tenant1.com` and map it to `adventureworks.z01.azurefd.net`.

TLS configuration

Adventure Works and its tenants need to decide who issues TLS certificates:

- The easiest option is to use Azure Front Door to issue and manage the certificates, but tenants shouldn't configure CCA records on their DNS servers. If they do, the records might prevent the Azure Front Door certification authority from issuing certificates.
- Alternatively, tenants can provide their own certificates. They need to work with Adventure Works to upload the certificate to a key vault and provide access to

Azure Front Door.

Azure Front Door configuration

One-time configuration: Adventure Works creates an Azure Front Door profile and a single endpoint. They configure an origin group for each stamp. They don't create custom domain resources or routes.

When a new tenant is onboarded: Adventure Works adds a custom domain resource to Azure Front Door. They use the tenant-provided domain name and associate the appropriate TLS certificate with the custom domain resource. They then create a route to specify which origin group (stamp) that tenant's requests should be directed to. In the preceding diagram, `tenant1app.tenant1.com` is routed to the origin group in the Australia region, and `tenant2app.tenant3.com` is routed to the origin group in the US region.

Benefits

- Customers can provide their own domain names. Azure Front Door transparently routes requests to the multitenant solution.
- Adventure Works maintains a single instance of Azure Front Door to route traffic to multiple stamps across multiple regions.

Drawbacks

- Adventure Works needs to reconfigure Azure Front Door every time a new tenant is onboarded.
- Tenants need to be involved in the onboarding process. They need to make DNS changes and possibly issue TLS certificates.
- Tenants control their DNS records. Changes to DNS records might affect their ability to access the Adventure Works solution.
- Adventure Works needs to pay attention to [Azure Front Door quotas and limits](#), especially on the number of routes and custom domains, and the [composite routing limit](#).

Scenario 5: Azure Front Door profile per stamp

You can deploy an Azure Front Door profile for each stamp. If you have 10 stamps, you deploy 10 instances of Azure Front Door. This approach can be useful if you need to restrict management access of each stamp's Azure Front Door configuration. It can also

be useful if you need to use multiple Azure Front Door profiles to avoid exceeding resource quotas or limits.

Tip

Azure Front Door is a global resource. Even if you deploy regionally scoped stamps, each Azure Front Door profile is globally distributed. You should consider whether you really need to deploy multiple Azure Front Door profiles, and what advantages you gain by doing so.

If you have a stamp that serves multiple tenants, you need to consider how you route traffic to each tenant. Review the approaches described in the preceding scenarios, and consider combining approaches to suit your requirements.

Benefits

- If you extend your configuration across multiple profiles, you're less likely to reach the Azure Front Door resource limits. For example, if you need to support high numbers of custom domains, you can divide the domains among multiple Azure Front Door profiles and stay within the limits of each profile.
- This approach enables you to scope your Azure Front Door resource management permissions. You can use Azure role-based access control (RBAC) to grant administrators access to a single stamp's profile.

Drawbacks

- This approach typically incurs a high cost because you deploy more profiles. For more information, see [Understand Front Door billing](#).
- There's a limit to the number of Azure Front Door profiles that you can deploy into a single Azure subscription. For more information, see [Front Door quotas and limits](#).
- You need to configure each stamp's Azure Front Door profile separately, and you need to manage DNS configuration, TLS certificates, and TLS configuration for each stamp.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Raj Nemani](#) | Director, Partner Technology Strategist
- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Mick Alberts](#) | Technical Writer
- [Fernando Antivero](#) | Fullstack Developer & Cloud Platform Engineer
- [Duong Au](#) | Senior Content Developer, C+E Skilling Content R&D
- [Harikrishnan M B \(HARI\)](#) | Product Manager 2, Azure Networking
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Training: Introduction to Azure Front Door](#)
- [Azure Front Door introduction](#)
- [What is Azure Front Door?](#)

Related resources

- [Architect multitenant solutions on Azure](#)
- [Checklist for architecting and building multitenant solutions on Azure](#)
- [Tenancy models to consider for a multitenant solution](#)

Azure NAT Gateway considerations for multitenancy

Article • 03/20/2023

Azure NAT Gateway provides control over outbound network connectivity from your resources that are hosted within an Azure virtual network. In this article, we review how NAT Gateway can mitigate Source Network Address Translation (SNAT) port exhaustion, which can affect multitenant applications. We also review how NAT Gateway assigns static IP addresses to the outbound traffic from your multitenant solution.

ⓘ Note

Firewalls, like [Azure Firewall](#), enable you to control and log your outbound traffic. Azure Firewall also provides similar SNAT port scale and outbound IP address control to NAT Gateway. NAT Gateway is less costly, but it also has fewer features and is not a security product.

Features of NAT Gateway that support multitenancy

High-scale SNAT ports

SNAT ports are allocated when your application makes multiple concurrent outbound connections to the same public IP address, on the same port. SNAT ports are a finite resource within [load balancers](#). If your application opens large numbers of separate connections to the same host, it can consume all of the available SNAT ports. This situation is called *SNAT port exhaustion*.

In most applications, SNAT port exhaustion indicates that your application is incorrectly handling HTTP connections or TCP ports. However, some multitenant applications are at particular risk of exceeding SNAT port limits, even if they reuse connections appropriately. For example, this situation can occur when your application connects to many tenant-specific databases behind the same database gateway.

ⓘ Tip

If you observe SNAT port exhaustion in a multitenant application, you should verify whether **your application follows good practices**. Ensure you reuse HTTP connections and don't recreate new connections every time you connect to an external service. You might be able to deploy a NAT Gateway to work around the problem, but if your code doesn't follow the best practices, you could encounter the problem again in the future.

The issue is exacerbated when you work with Azure services that share SNAT port allocations between multiple customers, such as [Azure App Service](#) and [Azure Functions](#).

If you determine you're experiencing SNAT exhaustion and are sure your application code correctly handles your outbound connections, consider deploying NAT Gateway. This approach is commonly used by customers who deploy multitenant solutions that are built on [Azure App Service](#) and [Azure Functions](#).

Each public IP address attached to NAT gateway provides 64,512 SNAT ports for connecting outbound to the internet. NAT gateway can scale to use up to 16 public IP addresses which provides over 1 million SNAT ports. If you need to scale beyond this limit, you can consider [deploying multiple NAT Gateway instances across multiple subnets or VNets](#). Each virtual machine in a subnet can use any of the available SNAT ports, if it needs them.

Outbound IP address control

Outbound IP address control can be useful in multitenant applications, when you have all of the following requirements:

- You use Azure services that don't automatically provide dedicated static IP addresses for outbound traffic. These services include Azure App Service, Azure Functions, API Management (when running in the consumption tier), and Azure Container Instances.
- You need to connect to your tenants' networks over the internet.
- Your tenants need to filter incoming traffic based on the IP address of each request.

When a NAT Gateway instance is applied to a subnet, any outbound traffic from that subnet uses the public IP addresses that's associated with the NAT gateway.

Note

When you associate multiple public IP addresses with a single NAT Gateway, your outbound traffic could come from any of those IP addresses. You might need to

configure firewall rules at the destination. You should either allow each IP address, or use a [public IP address prefix](#) resource to use a set of public IP addresses in the same range.

Isolation models

If you need to provide different outbound public IP addresses for each tenant, you must deploy individual NAT Gateway resources. Each subnet can be associated with a single NAT Gateway instance. To deploy more NAT gateways, you need to deploy multiple subnets or virtual networks. In turn, you likely need to deploy multiple sets of compute resources.

Review [Architectural approaches for networking in multitenant solutions](#) for more information about how to design a multitenant network topology.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Aimee Littleton](#) | Program Manager 2, Azure NAT Gateway
- [Arsen Vladimirsingh](#) | Principal Customer Engineer, FastTrack for Azure
- [Joshua Waddell](#) | Senior Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Learn more about [NAT Gateway](#).
- Learn how to use [NAT Gateway with Azure App Service and Azure Functions](#).
- Review [Architectural approaches for networking in multitenant solutions](#).

Multitenancy and Azure Private Link

Article • 05/11/2023

Azure Private Link provides private IP addressing for Azure platform services, and for your own applications that are hosted on Azure virtual machines. You can use Private Link to enable private connectivity from your tenants' Azure environments. Tenants can also use Private Link to access your solution from their on-premises environments, when they're connected through virtual private network gateways (VPN Gateway) or ExpressRoute.

Azure Private Link is used by many large SaaS providers, including [Snowflake](#), [Confluent Cloud](#), and [MongoDB Atlas](#).

In this article, we review how you can configure Private Link for an Azure-hosted multitenant solution.

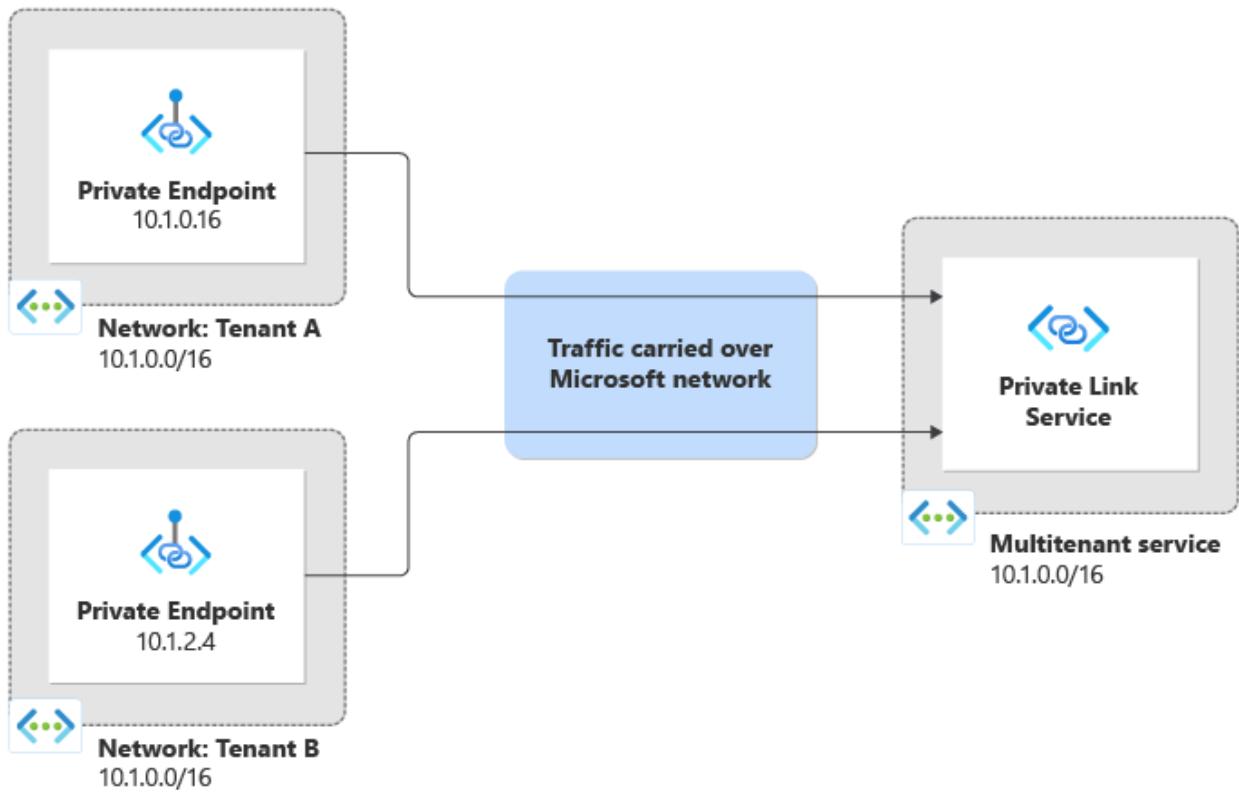
Key considerations

Overlapping IP address spaces

Private Link provides powerful capabilities for multitenant solutions, where tenants can access the service through private address spaces.

Different tenants frequently use the same or overlapping private IP address spaces. For example, your multitenant solution might use the IP address space of `10.1.0.0/16`. Suppose tenant A uses their own on-premises network with the same IP address space, and coincidentally tenant B also uses the same IP address space. You can't directly connect or peer your networks together because the IP address ranges overlap.

When you use Private Link to enable connectivity from each tenant to the multitenant solution, each tenant's traffic automatically has network address translation (NAT) applied. Each tenant can use a private IP address within their own respective network, and the traffic flows to the multitenant solution transparently. Private Link performs NAT on traffic, even when tenants and the service provider all use overlapping IP address ranges:



When traffic arrives into the multitenant solution, it has already been translated. This means traffic appears to originate from within the multitenant service's own virtual network IP address space. Private Link provides the [TCP Proxy Protocol v2](#) feature, which enables a multitenant service to know the tenant that sent the request, and even the original IP address from the source network.

Service selection

When you use Private Link, it's important to consider the service that you want to allow inbound connectivity to.

Azure Private Link service is used with virtual machines behind a standard load balancer.

You can also use Private Link with other Azure services. These services include application hosting platforms like Azure App Service. They also include Azure Application Gateway or Azure API Management, which are network and API gateways.

The application platform you use determines many aspects of your Private Link configuration, and the limits that apply. Additionally, some services don't support Private Link for inbound traffic.

Limits

Carefully consider the number of private endpoints that you can create, based on your solution's architecture. If you use a platform as a service (PaaS) application platform, it's

important to be aware of the maximum number of private endpoints that a single resource can support. If you run virtual machines, you can attach a Private Link service instance to a standard load balancer (SLB). In this configuration, you can generally connect a higher number of private endpoints, but limits still apply. These limits might determine how many tenants you can connect to your resources by using Private Link. Review [Azure subscription and service limits, quotas, and constraints](#) to understand the limits to the number of endpoints and connections.

Additionally, some services require a specialized networking configuration to use Private Link. For example, if you use Private Link with Azure Application Gateway, you must [provision a dedicated subnet](#), in addition to the standard subnet for the Application Gateway resource.

Carefully test your solution, including your deployment and diagnostic configuration, with your Private Link configuration enabled. Some Azure services block public internet traffic, when a private endpoint is enabled, which can require that you change your deployment and management processes.

Private Link in combination with public-facing services

You might choose to deploy your solution to be both internet-facing and also to be exposed through private endpoints. Consider your overall network topology and the paths that each tenant's traffic follows.

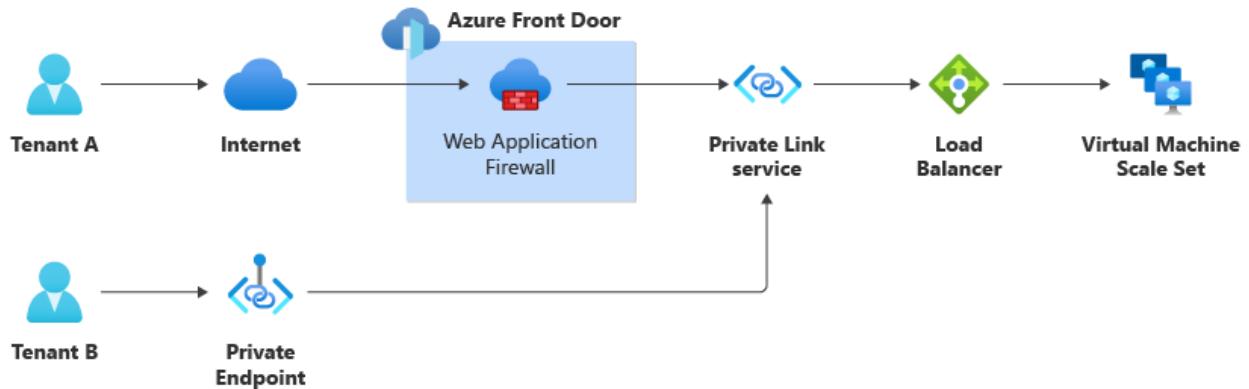
When your solution is based on virtual machines that are behind a standard load balancer, you can expose your endpoint via the Private Link service. In this case, a web application firewall and application routing are likely already part of your virtual machine-based workload.

Many Azure PaaS services support Private Link for inbound connectivity, even across different Azure subscriptions and Azure Active Directory tenants. You can use that service's Private Link capabilities to expose your endpoint.

When you use other internet-facing services, like Azure Front Door, it's important to consider whether they support Private Link for inbound traffic. If they don't, consider how your traffic flows through each path to your solution.

For example, suppose you build an internet-facing application that runs on a virtual machine scale set. You use Azure Front Door, including its web application firewall (WAF), for security and traffic acceleration, and you [configure Front Door to send its traffic through a private endpoint to your backend \(origin\) service](#). Tenant A connects to your solution by using a public endpoint, and tenant B connects by using a private

endpoint. Because Front Door doesn't support Private Link for incoming connections, tenant B's traffic bypasses your Front Door and its WAF:



Isolation models

Private Link is designed to support scenarios where a single application tier can be used by multiple separate clients, such as your tenants. When you consider isolation for Private Link, the main concern is around the number of resources you need to deploy to support your requirements. The tenant isolation models you can use for Private Link depend on the service that you use.

Isolation models for Private Link service

If you use Private Link service with virtual machines behind a standard load balancer, there are several isolation models that you can consider.

Consideration	Shared Private Link service and shared load balancer	Dedicated Private Link service and dedicated load balancer	Dedicated Private Link service and shared load balancer
Deployment complexity	Low	Medium-high, depending on the number of tenants	Medium-high, depending on the number of tenants
Operational complexity	Low	Medium-high, depending on the number of resources	Medium-high, depending on the number of resources
Limits to consider	Number of private endpoints on the same Private Link service	Number of Private Link services per subscription	Number of Private Link services per standard load balancer
Example scenario	Large multitenant solution with shared application tier	Separate deployment stamps for each tenant	Shared application tier in a single stamp, with large numbers of tenants

In all three models, the level of data isolation and performance depends on the other elements of your solution, and the Private Link service deployment doesn't materially affect these factors.

Shared Private Link service and shared standard load balancer

You might consider deploying a shared Private Link service, which is connected to a standard load balancer. Each of your tenants can create a private endpoint and use it to connect to your solution.

A single Private Link service instance supports a large number of private endpoints. If you exhaust the limit, you can deploy more Private Link service instances, although there are also limits to the number of Private Link services you can deploy on a single load balancer. If you expect that you'll approach these limits, consider using a Deployment Stamps-based approach, and deploy shared load balancers and Private Link service instances into each stamp.

Dedicated Private Link service and dedicated standard load balancer per tenant

You can deploy a dedicated Private Link service and dedicated load balancer for each tenant. This approach makes sense when you have a dedicated set of virtual machines for each tenant, such as when your tenants have strict compliance requirements.

Dedicated Private Link service per tenant and shared standard load balancer

You can also deploy dedicated Private Link service instances for each tenant, with a shared standard load balancer. However, this model is unlikely to provide much benefit. Additionally, because there's a limit to the number of Private Link services that you can deploy on a single standard load balancer, this model isn't likely to scale beyond a small multitenant solution.

More commonly, you can deploy multiple shared Private Link services. This approach enables you to expand the number of private endpoints that your solution can support on one shared load balancer.

Isolation models for Azure PaaS services with private endpoints

When you deploy Azure platform as a service (PaaS) services and want to enable tenants to access those services with private endpoints, then you need to consider the capabilities and constraints of the specific service. Additionally, you need to consider whether your application tier resources are dedicated to a specific tenant or if they're shared between tenants.

If you deploy a dedicated set of application tier resources for each tenant, it's likely that you can deploy one private endpoint for that tenant to use to access their resources. It's unlikely that you'll exhaust any Private Link-related service limits, because each tenant has their own resources dedicated to them.

When you share application tier resources between tenants, you might consider deploying a private endpoint for each tenant. There are limits on the number of private endpoints that can be attached to a single resource, and these limits are different for each service.

Features of Azure Private Link that support multitenancy

Private Link has several features that are helpful in a multitenant environment. However, the specific features available to you depend on the service you use. The foundational Azure Private Link service, for virtual machines and load balancers, supports all of the features described below. Other services with Private Link support might provide only a subset of these features.

Service aliases

When a tenant configures access to your service by using Private Link, they need to be able to identify your service so that Azure can establish the connection.

Private Link service, and certain other Private Link-compatible Azure services, enable you to [configure an alias](#) that you provide to your tenants. By using an alias, you avoid disclosing your Azure subscription IDs and resource group names.

Service visibility

The Private Link service enables you to [control the visibility of your private endpoint](#). You might allow all Azure customers to connect to your service, if they know its alias or resource ID. Alternatively, you might restrict access to just a set of known Azure customers.

You can also specify a limited number of pre-approved Azure subscription IDs that can connect to your private endpoint. If you choose to use this approach, consider how you'll collect and authorize subscription IDs. For example, you might provide an administration user interface in your application to collect a tenant's subscription ID. Then, you can dynamically reconfigure your Private Link service instance to pre-approve that subscription ID for connections.

Connection approvals

After a connection has been requested between a client (like a tenant) and a private endpoint, Private Link requires that the connection is *approved*. Until the connection is approved, traffic can't flow through the private endpoint connection.

The Private Link service supports several types of approval flows, including:

- **Manual approval**, where your team explicitly approves every connection. This approach is viable when you have only a few tenants who use your service through Private Link.
- **API-based approval**, where the Private Link service treats the connection as requiring a manual approval, and your application uses the [Update Private Endpoint Connection API](#), the Azure CLI, or Azure PowerShell to approve a connection. This approach can be useful when you have a list of tenants who have been authorized to use private endpoints.
- **Auto-approval**, where the Private Link service itself maintains the list of subscription IDs that should have their connections automatically approved.

For more information, see [Control service access](#).

Proxy Protocol v2

When you use the Private Link service, by default your application only has visibility of an IP address that has been through network address translation (NAT). This behavior means that traffic appears to flow from within your own virtual network.

Private Link enables you to get access to the original client IP address, in the tenant's virtual network. This feature uses the [TCP Proxy Protocol v2](#).

For example, suppose your tenants' administrators need to add IP address-based access restrictions, such as *host 10.0.0.10 can access the service, but host 10.0.0.20 can't*. When you use Proxy Protocol v2, you can enable your tenants to configure these types of access restrictions in your application. However, your application code needs to inspect the client's original IP address and enforce the restrictions.

Related resources

- [Azure Private Link Service explanation and demos from provider \(SaaS ISV\) and consumer perspectives](#) : A video that looks at the Azure Private Link service feature that enables multitenant service providers (such as independent software vendors building SaaS products). This solution enables consumers to access the provider's service using private IP addresses from the consumer's own Azure virtual networks.
- [TCP Proxy Protocol v2 with Azure Private Link Service—Deep Dive](#) : A video that presents a deep dive into TCP Proxy Protocol v2, which is an advanced feature of the Azure Private Link service. It's useful in multitenant and SaaS scenarios. The video shows you how to enable Proxy Protocol v2 in the Azure Private Link service. It also shows you how to configure an NGINX service to read the source private IP address of the original client, rather than the NAT IP, to access the service via the private endpoint.
- [Using NGINX Plus to decode Proxy Protocol TLV linkIdentifier from the Azure Private Link service](#) : A video that looks at how to use NGINX Plus to get the TCP Proxy Protocol v2 TLV from the Azure Private Link service. The video shows how you can then extract and decode the numeric `linkIdentifier`, also called `LINKID`, of the private endpoint connection. This solution is useful for multitenant providers who need to identify the specific consumer tenant from which the connection was made.
- [SaaS Private Connectivity pattern](#) : An example solution that illustrates one approach to automate the approval of private endpoint connections, by using Azure Managed Applications.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirsksiy](#) | Principal Customer Engineer, FastTrack for Azure

Other contributor:

- [Sumeet Mittal](#) | Principal Product Manager, Azure Private Link

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review the [networking approaches for multitenancy](#).

Multitenancy and Azure Cache for Redis

Article • 05/18/2023

Azure Cache for Redis is commonly used to increase the performance of your solution, to reduce the load on your database or other data-tier components, and to reduce the amount of state that's stored on compute nodes. In this article, we describe some of the features of Azure Cache for Redis that are useful for multitenant solutions, and then we provide links to the guidance that can help you, when you're planning how you're going to use Azure Cache for Redis.

Isolation models

When working with a multitenant system that uses Azure Cache for Redis, you need to make a decision about the level of isolation you want to use. Azure Cache for Redis supports several isolation models.

The following table summarizes the differences between the main tenancy isolation models for Azure Cache for Redis:

Consideration	Shared cache, shared database	Shared cache, database per tenant	Cache per tenant
Data isolation	Low. Use Redis data structures or key prefixes to identify each tenant's data	Low. Data is separated but no security isolation is provided	High
Performance isolation	Low. All tenants share the same compute resources	Low. All tenants share the same compute resources	High
Deployment complexity	Low	Medium	Medium-high
Operational complexity	Low	Low	Medium-high
Resource cost	Low	Low	High
Example scenario	Large multitenant solution with a shared application tier	Migrating a single-tenant application to be multitenant-aware	Individual application instances per tenant

Shared cache instance and shared database

You might consider deploying a single cache, with a single Redis database, and using it to store cached data for all of your tenants. This approach is commonly used when you have a single application tier that all of your tenants share.

To isolate tenant-specific data within each cache, consider using key prefixes to prepend the tenant ID. Your application can then access specific data for a specific tenant.

Alternatively, you can consider using Redis data structures, like [sets](#) or [hashes](#), for each tenant's data. Both sets and hashes support a large number of keys, so this approach can scale to many tenants.

When you use a single cache instance, the application needs to be authorized to access the entire cache. Azure Cache for Redis doesn't provide granular access control within a cache.

When you use this approach, consider that all of your tenants will share the same underlying compute resources for the cache. So, this approach can be vulnerable to the [Noisy Neighbor problem](#). Ensure that you follow the best practices for Azure Cache for Redis for [scaling, memory management](#), and [server load](#), to make the most efficient use of your cache's resources and to mitigate any noisy neighbor effects.

Additionally, consider monitoring your cache's resources, such as CPU and memory. If you observe resource pressure, consider the following mitigations:

- Scale up to a cache SKU or tier with higher levels of resources.
- Scale out to multiple caches by sharding your cached data. You can either shard by tenant, where some tenants use cache A and some use cache B, or you can shard by subsystem, where one part of your solution caches data for all tenants to cache A, and another part of your solution caches onto cache B.

Shared cache instance with a database per tenant

Another approach you might consider is to deploy a single cache instance, and deploy tenant-specific Redis databases within the instance. This approach provides some degree of logical isolation of each tenant's data, but it doesn't provide any performance isolation or protection against noisy neighbors.

This approach might be useful for migration scenarios. For example, suppose you modernize a single-tenant application that isn't designed to work with multiple tenants, and you gradually convert it to be multitenancy-aware by including the tenant context in all requests. You can gain some cost efficiencies by using a single shared cache, and you won't need to update the application's logic to use tenant key prefixes or tenant-specific data structures.

Azure Cache for Redis imposes [limits on the number of databases that can be created on a single cache](#). Ensure that you understand the number of tenants that you'll grow to, before you implement this approach.

Additionally, this approach doesn't provide any benefits for security isolation of data. In Azure Cache for Redis, authentication and authorization are performed at the cache instance level.

 Note

Azure Cache for Redis supports multiple databases on specific tiers, and it doesn't support multiple databases when you use clustered instances.

Cache instance per tenant

You might consider deploying a separate instance of Azure Cache for Redis for each tenant. There's no limit to the number of caches you can deploy within a single Azure subscription. This approach provides the strongest level of data and performance isolation.

However, each cache is billed as a separate Azure resource, so as you grow to large numbers of tenants, you might incur more cost. Furthermore, this approach often doesn't make efficient use of each cache's resources, since each Azure Cache for Redis instance generally supports large volumes of requests. It's best to only consider this isolation approach if you have strict data or performance isolation requirements.

Features of Azure Cache for Redis that support multitenancy

Active geo-replication

Many multitenant solutions need to be geo-distributed. You might share a globally distributed application tier, with your application instances reading from and writing to a nearby cache to maintain acceptable performance. The Enterprise tier of Azure Cache for Redis supports [linking multiple caches together across regions, in an active-active configuration](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Will Velida](#) | Customer Engineer 2, FastTrack for Azure

Other contributors:

- [Carl Dacosta](#) | Principal Software Engineering Manager, Azure Cache for Redis
- [Kyle Teegarden](#) | Senior Program Manager, Azure Cache for Redis
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [storage and data approaches for multitenancy](#).

Multitenancy and Azure Cosmos DB

Article • 07/10/2023

On this page, we describe some of the features of Azure Cosmos DB that are useful when you're working with multitenant systems. We also link to guidance and examples for how to use Azure Cosmos DB in a multitenant solution.

Features of Azure Cosmos DB that support multitenancy

Partitioning

By using partitions with your Azure Cosmos DB containers, you can create containers that are shared across multiple tenants. Typically you use the tenant identifier as a partition key, but you might also consider using multiple partition keys for a single tenant. A well-planned partitioning strategy effectively implements the [Sharding pattern](#). With large containers, Azure Cosmos DB spreads your tenants across multiple physical nodes to achieve a high degree of scale.

We recommend that you explore the use of [hierarchical partition keys](#) to improve the performance of your multitenant solution. Hierarchical partition keys enable you to create a partition key that includes multiple values. For example, you might use a hierarchical partition key that includes the tenant identifier and the type of data that you're storing. This approach allows you to scale beyond the logical partition limit of 20 GB per partition key value.

More information:

- [Partitioning and horizontal scaling in Azure Cosmos DB](#)
- [Hierarchical partition keys](#)

Managing request units

Azure Cosmos DB pricing model is based on the number of *request units* per second that you provision or consume. A request unit is a logical abstraction of the cost of a database operation or query. Typically, you provision a defined number of request units per second for your workload, which is referred to as *throughput*. Azure Cosmos DB provides several options for how you provision throughput. In a multitenant

environment, the selection you make affects the performance and price of your Azure Cosmos DB resources.

One tenancy model for Azure Cosmos DB involves deploying separate containers for each tenant within a shared database. Azure Cosmos DB enables you to provision request units for a database, and all of the containers share the request units. If your tenant workloads don't typically overlap, this approach can help reduce your operational costs. However, this approach is susceptible to the [Noisy Neighbor problem](#) because a single tenant's container might consume a disproportionate amount of the shared provisioned request units. To mitigate this issue, first identify the noisy tenants. Then, you can optionally set provisioned throughput on a specific container. The other containers in the database continue to share their throughput, but the noisy tenant consumes their own dedicated throughput.

Azure Cosmos DB also provides a serverless tier, which is suited for workloads with intermittent or unpredictable traffic. Alternatively, autoscaling enables you to configure policies to specify the scaling of provisioned throughput. Additionally, you can take advantage of Azure Cosmos DB burst capacity, maximizing the utilization of your provisioned throughput capacity, which would have been rate limited otherwise. In a multitenant solution, you might combine all of these approaches to support different types of tenant.

 **Note**

When planning your Azure Cosmos DB configuration, ensure you consider the [service quotas and limits](#).

To monitor and manage the costs that are associated with each tenant, every operation using the Azure Cosmos DB API includes the request units consumed. You can use this information to aggregate and compare the actual request units consumed by each tenant, and you can then identify tenants with different performance characteristics.

More information:

- [Provisioned throughput](#)
- [Autoscale](#)
- [Serverless](#)
- [Measuring the RU charge of a request](#)
- [Azure Cosmos DB service quotas](#)
- [Burst capacity](#)

Customer-managed keys

Some tenants might require the use of their own encryption keys. Azure Cosmos DB provides a customer-managed key feature. This feature is applied at the level of an Azure Cosmos DB account, so tenants who require their own encryption keys need to be deployed using dedicated Azure Cosmos DB accounts.

More information:

- [Configure customer-managed keys for your Azure Cosmos DB account with Azure Key Vault](#)

Isolation models

When working with a multitenant system that uses Azure Cosmos DB, you need to make a decision about the level of isolation you want to use. Business-to-business (B2B) refers to selling to a business. Business-to-consumer (B2C) refers to selling directly to an individual customer who uses the product or service. Azure Cosmos DB supports several isolation models:

	Partition key per tenant	Container per tenant (shared throughput)	Container per tenant (dedicated throughput)	Database per tenant	Database account per tenant
Queries across tenants	Easy (container acts as boundary for queries)	Hard	Hard	Hard	Hard
Tenant density	High (lowest cost per tenant)	Medium	Low	Low	Low
Tenant data deletion	Hard	Easy (drop container when tenant leaves)	Easy (drop container when tenant leaves)	Easy (drop database when tenant leaves)	Easy (drop database when tenant leaves)
Data access security isolation	Needs to be implemented within the application	Container RBAC	Container RBAC	Database RBAC	RBAC
Geo-replication	Per tenant geo-	Group tenants within	Group tenants within	Group tenants within	Group tenants within

	Partition key not possible per tenant	Database accounts based on tenant (shared throughput)	Database accounts based on tenant (dedicated throughput)	Database per tenant based on requirements	Database accounts based on tenant (per account)
Noisy neighbor prevention	None	None	Yes	Yes	Yes
New tenant creation latency	Immediate	Fast	Fast	Medium	Slow
Data modeling advantages	None	entity colocation	entity colocation	Multiple containers to model tenant entities	Multiple containers and databases to model tenants
Encryption key	Same for all tenants	Same for all tenants	Same for all tenants	Same for all tenants	Customer managed key per tenant
Throughput requirements	>0 RUs per tenant	>100 RUs per tenant	>100 RUs per tenant (with autoscale only, otherwise >400 RUs per tenant)	>400 RUs per tenant	>400 RUs per tenant
Example use case(s)	B2C apps	Standard offer for B2B apps	Premium offer for B2B apps	Premium offer for B2B apps	Premium offer for B2B apps

Partition key per tenant

When you use a single container for multiple tenants, you can make use of Azure Cosmos DB partitioning support. By using separate partition keys for each tenant, you can easily query the data for a single tenant. You can also query across multiple tenants, even if they are in separate partitions. However, [cross-partition queries](#) have a higher request unit (RU) cost than single-partition queries.

This approach tends to work well when the amount of data stored for each tenant is small. It can be a good choice for building a [pricing model](#) that includes a free tier, and for business-to-consumer (B2C) solutions. In general, by using shared containers, you achieve the highest density of tenants and therefore the lowest price per tenant.

It's important to consider the throughput of your container. All of the tenants will share the container's throughput, so the [Noisy Neighbor problem](#) can cause performance challenges if your tenants have high or overlapping workloads. This problem can sometimes be mitigated by grouping subsets of tenants into different containers, and by ensuring that each tenant group has compatible usage patterns. Alternatively, you can consider a hybrid multi- and single-tenant model. Group smaller tenants into shared partitioned containers, and give large customers dedicated containers. Also, there are features that can help control the noisy neighbor problem when modeling tenant by partition, such as [throughput reallocation](#), [burst capacity](#), and [throughput control](#) in the [Java SDK](#).

It's also important to consider the amount of data you store in each logical partition. Azure Cosmos DB allows each logical partition to grow to up to 20 GB. If you have a single tenant that needs to store more than 20 GB of data, consider spreading the data across multiple logical partitions. For example, instead of having a single partition key of `Contoso`, you might *salt* the partition keys by creating multiple partition keys for a tenant, such as `Contoso1`, `Contoso2`, and so forth. When you query the data for a tenant, you can use the `WHERE IN` clause to match all of the partition keys. [Hierarchical partition keys](#) can also be used to support large tenants, with storage greater than 20 GB, without having to use synthetic partition keys or multiple partition key values per tenant.

Consider the operational aspects of your solution, and the different phases of the [tenant lifecycle](#). For example, when a tenant moves to a dedicated pricing tier, you'll likely need to move the data to a different container. When a tenant is deprovisioned, you need to run a delete query on the container to remove the data, and for large tenants, this query might consume a significant amount of throughput while it executes.

Container per tenant

You can provision dedicated containers for each tenant. Dedicated containers work well when the data that you store for your tenant can be combined into a single container. This model provides greater performance isolation than the partition-key-per-tenant model above, and it also provides increased data access security isolation via [Azure RBAC](#).

When using a container for each tenant, you can consider sharing throughput with other tenants by provisioning throughput at the database level. Consider the restrictions and limits around the [minimum number of request units for your database](#) and the [maximum number of containers in the database](#). Also, consider whether your tenants require a guaranteed level of performance, and whether they're susceptible to the [Noisy Neighbor problem](#). When you share throughput at the database level, the workload or

storage across all the containers should be relatively uniform. Otherwise you might have a noisy neighbor issue, if there are one or more large tenants. If necessary, plan to group these tenants into different databases that are based on workload patterns.

Alternatively, you can provision dedicated throughput for each container. This approach works well for larger tenants and for tenants that are at risk of the [Noisy Neighbor problem](#). However, the baseline throughput for each tenant is higher, so consider the minimum requirements and cost implications of this model.

If your tenant data model requires more than one entity, as long as all entities can share the same partition key, they can be colocated in the same container. However, if the tenant data model is more complex, and it requires entities that can't share the same partition key, consider the database-per-tenant or database-account-per-tenant models below. Take a look at our article on [how to model and partition data on Azure Cosmos DB using a real-world example](#) for more guidance.

Lifecycle management is generally simpler when containers are dedicated to tenants. You can [easily move tenants between shared and dedicated throughput models](#), and when deprovisioning a tenant, you can quickly delete the container.

Database per tenant

You can provision databases for each tenant, in the same database account. Like the container-per-tenant model above, this model provides greater performance isolation than the partition-key-per-tenant model, and it also provides increased data access security isolation via [Azure RBAC](#).

Like the account-per-tenant model below, this approach gives the highest level of performance isolation, but it provides the lowest tenant density. However, this option is best when each tenant requires a more complicated data model than is feasible in the container-per-tenant model. Or, you should follow this approach when it's a requirement for new tenant creation to be fast and/or free of any overhead to create tenant accounts up-front. It may also be the case, for the specific software development framework being used, that database-per-tenant is the only level of performance isolation that's recognized in that framework. Entity (container) level isolation and entity colocation aren't typically supported natively in such frameworks.

Database account per tenant

Azure Cosmos DB enables you to provision separate database accounts for each tenant, which provides the highest level of isolation, but the lowest density. Like the container-per-tenant and database-per-tenant models above, this model provides greater

performance isolation than the partition-key-per-tenant key model. It also provides increased data access security isolation via [Azure RBAC](#). In addition, this model provides database encryption security isolation at the tenant level via [customer managed keys](#).

A single database account is dedicated to a tenant, which means they aren't subject to the noisy neighbor problem. You can configure the location of the database account according to the tenant's requirements. You can also tune the configuration of Azure Cosmos DB features, such as geo-replication and customer-managed encryption keys, to suit each tenant's requirements. When using a dedicated Azure Cosmos DB account per tenant, consider the [maximum number of Azure Cosmos DB accounts per Azure subscription](#).

If you use this model, you should consider how fast your application needs to be able to generate new tenants. Account creation in Azure Cosmos DB can take a few minutes, so you might need to create accounts up-front. If this approach isn't feasible, consider the database-per-tenant model.

If you allow tenants to migrate from a shared account to a dedicated Azure Cosmos DB account, consider the migration approach you'll use to move a tenant's data between the old and new accounts.

Hybrid approaches

You can consider a combination of the above approaches to suit different tenants' requirements and [your pricing model](#). For example:

- Provision all free trial customers within a shared container, and use the tenant ID or a [synthetic key partition key](#).
- Offer a paid *Bronze* tier that deploys a dedicated container per tenant, but with [shared throughput on a database](#).
- Offer a higher *Silver* tier that provisions dedicated throughput for the tenant's container.
- Offer the highest *Gold* tier, and provide a dedicated database account for the tenant, which also allows tenants to select the geography for their deployment.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Paul Burpo](#) | Principal Customer Engineer, FastTrack for Azure
- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Mark Brown](#) | Principal PM Manager, Azure Cosmos DB
- [Deborah Chen](#) | Principal Program Manager
- [Theo van Kraay](#) | Senior Program Manager, Azure Cosmos DB
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure
- Thomas Weiss | Principal Program Manager
- [Vic Perdana](#) | Cloud Solution Architect, Azure ISV

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [storage and data approaches for multitenancy](#).

Learn more about multitenancy and Azure Cosmos DB:

- [Azure Cosmos DB and multitenant systems](#): A blog post that discusses how to build a multitenant system that uses Azure Cosmos DB.
- [Multitenant applications with Azure Cosmos DB](#) (video)
- [Building a multitenant SaaS with Azure Cosmos DB and Azure](#) (video): A real-world case study of how Whally, a multitenant SaaS startup, built a modern platform from scratch on Azure Cosmos DB and Azure. Whally shows the design and implementation decisions they made that relate to partitioning, data modeling, secure multitenancy, performance, real-time streaming from change feed to SignalR, and more. All these solutions use ASP.NET Core on Azure App Services.

Related resources

Refer to some of our other Cosmos DB architectural scenarios:

- [Multi-region web application with Azure Cosmos DB replication](#)
- [Globally distributed applications using Azure Cosmos DB](#)
- [Serverless apps using Azure Cosmos DB](#)
- [Azure Cosmos DB in IoT workloads](#)
- [Transactional Outbox pattern with Azure Cosmos DB](#)
- [Scalable order processing](#)
- [Visual search in retail with Azure Cosmos DB](#)

- Personalization using Azure Cosmos DB
- Gaming using Azure Cosmos DB

Multitenancy and Azure Database for PostgreSQL

Article • 07/07/2023

Many multitenant solutions on Azure use the open-source relational database management system Azure Database for PostgreSQL. In this article, we review the features of Azure Database for PostgreSQL that are useful when working with multitenant systems. The article also links to guidance and examples for how to use Azure Database for PostgreSQL, in a multitenant solution.

Deployment modes

There are two deployment modes available for Azure Database for PostgreSQL that are suitable for use with multitenant applications:

- [Flexible Server](#) - This is a good choice for most multitenant deployments that don't require the high scalability that's provided by Azure Cosmos DB for PostgreSQL.
- [Azure Cosmos DB for PostgreSQL](#) - An Azure managed database service designed for solutions requiring a high level of scale, which often includes multitenanted applications. This service is part of the Azure Cosmos DB family of products.

Note

Azure Database for PostgreSQL - Single Server is on the retirement path and is [scheduled for retirement by March 28, 2025](#). It is not recommended for new multitenant workloads.

Features of Azure Database for PostgreSQL that support multitenancy

When you're building a multitenant application using Azure Database for PostgreSQL, there are a number of features that you can use to enhance the solution.

Note

Some features are only available in specific [deployment modes](#). These features are indicated in the guidance below.

Row-level security

Row-level security is useful for enforcing tenant-level isolation, when you use shared tables. In PostgreSQL, row-level security is implemented by applying *row security policies* to tables to restrict access to rows by tenant.

There maybe a slight performance impact when implementing row-level security on a table. Therefore, additional indexes might need to be created on tables with row-level security enabled to ensure performance is not impacted. It is recommended to use performance testing techniques to validate that your workload meets your baseline performance requirements when row-level security is enabled.

More information:

- [Azure Database for PostgreSQL Flexible Server row-level security](#)

Horizontal scaling with sharding

The [Sharding pattern](#) enables you to scale your workload across multiple databases or database servers.

Solutions that need a very high level of scale can use Azure Cosmos DB for PostgreSQL. This deployment mode enables horizontal sharding of tenants across multiple servers (nodes). By using *distributed tables* in multitenant databases, you can ensure all data for a tenant is stored on the same node, which increases query performance.

 **Note**

From October 2022, Azure Database for PostgreSQL Hyperscale (Citus) has been rebranded as Azure Cosmos DB for PostgreSQL and [moved into the Cosmos DB family of products](#).

More information:

- [Design a multi-tenant database using Azure Cosmos DB for PostgreSQL](#)
- [Distributed tables](#)
- Choosing a [distribution column](#) in a distributed table.
- A guide to using [Citus for multitenant applications](#) ↗.

Connection pooling

Postgres uses a process-based model for connections. This model makes it inefficient to maintain large numbers of idle connections. Some multitenant architectures require a large number of active connections, which will negatively impact the performance of the Postgres server.

Connection pooling via PgBouncer is installed by default in Azure Database for PostgreSQL [Flexible Server](#).

More information:

- [PgBouncer in Azure Database for PostgreSQL - Flexible Server](#)
- [Connection pooling in Azure Cosmos DB for PostgreSQL](#)
- [Steps to install and set up PgBouncer connection pooling proxy with Azure Database for PostgreSQL ↗](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Daniel Scott-Raynsford ↗](#) | Partner Technology Strategist

Other contributors:

- [John Downs ↗](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirskiy ↗](#) | Principal Customer Engineer, FastTrack for Azure
- [Paul Burpo ↗](#) | Principal Customer Engineer, FastTrack for Azure ISVs
- [Assaf Fraenkel ↗](#) | Senior Engineer/Data Architect, Azure FastTrack for ISVs and Start-ups

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [storage and data approaches for multitenancy](#).

Multitenancy and Azure SQL Database

Article • 12/20/2023

Multitenant solutions on Azure commonly use Azure SQL Database. On this page, we describe some of the features of Azure SQL Database that are useful when you design a multitenant system. We also link to guidance and examples for how to use Azure SQL in a multitenant solution.

Guidance

The Azure SQL Database team publishes extensive guidance on implementing multitenant architectures with Azure SQL Database. See [Multi-tenant SaaS patterns with Azure SQL Database](#). Also, consider the guidance for [partitioning Azure SQL databases](#).

Features of Azure SQL Database that support multitenancy

Azure SQL Database includes many features that support multitenancy.

Elastic pools

Elastic pools enable you to share compute resources between many databases on the same server. By using elastic pools, you can achieve performance elasticity for each database, while also achieving cost efficiency by sharing your provisioned resources across databases. Elastic pools provide built-in protections against the [Noisy Neighbor problem](#).

More information:

- [SQL Database elastic pools](#)
- [Resource management in dense elastic pools](#)
- [Disaster recovery strategies for applications using SQL Database elastic pools](#)

Elastic database tools

The [Sharding pattern](#) enables you to scale your workload across multiple databases. Azure SQL Database provides tools to support sharding. These tools include the management of *shard maps* (a database that tracks the tenants assigned to each shard).

They also include initiating and tracking queries and management operations on multiple shards by using *elastic jobs*.

More information:

- [Multi-tenant applications with elastic database tools and row-level security](#)
- [Scaling out with Azure SQL Database](#)
- [Elastic database jobs](#)
- The [Elastic Jobs tutorial](#) describes the process of creating, configuring, and managing elastic jobs.

Row-level security

Row-level security is useful for enforcing tenant-level isolation, when you use shared tables.

More information:

- [Video overview ↗](#)
- [Documentation](#)
- [Multi-tenant applications with elastic database tools and row-level security](#)

Key management

The Always Encrypted feature provides the end-to-end encryption of your databases. If your tenants require they supply their own encryption keys, consider deploying separate databases for each tenant and consider enabling the Always Encrypted feature.

More information:

- [Always Encrypted](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Paul Burpo ↗](#) | Principal Customer Engineer, FastTrack for Azure
- [John Downs ↗](#) | Principal Program Manager

Other contributors:

- [Silvano Coriani](#) | Principal Program Manager, Azure SQL
- [Dimitri Furman](#) | Principal Program Manager, Azure SQL
- [Sanjay Mishra](#) | Principal Group Program Manager, Azure SQL
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [storage and data approaches for multitenancy](#).

Related resources

- [Data partitioning strategies for Azure SQL Database](#)
- [Case study: Running 1M databases on Azure SQL for a large SaaS provider: Microsoft Dynamics 365 and Power Platform](#)
- [Sample: The Wingtip Tickets SaaS application](#) provides three multitenant examples of the same app; each explores a different database tenancy pattern on Azure SQL Database. The first uses a standalone application, per tenant with its own database. The second uses a multitenant app with a database, per tenant. The third sample uses a multitenant app with sharded multitenant databases.
- [Video: Multitenant design patterns for SaaS applications on Azure SQL Database](#)

Multitenancy and Azure Storage

Article • 07/10/2023

Azure Storage is a foundational service used in almost every solution. Multitenant solutions often use Azure Storage for blob, file, queue, and table storage. On this page, we describe some of the features of Azure Storage that are useful for multitenant solutions, and then we provide links to the guidance that can help you, when you're planning how you're going to use Azure Storage.

Features of Azure Storage that support multitenancy

Azure Storage includes many features that support multitenancy.

Shared access signatures

When you work with Azure Storage from a client application, it's important to consider whether client requests should be sent through another component that you control, like a content delivery network or API, or if the client should connect directly to your storage account. There might be good reasons to send requests through another component, including caching data at the edge of your network. However, in some situations, it's advantageous for client endpoints to connect directly to Azure Storage to download or upload data. This connection helps you improve the performance of your solution, especially when you work with large blobs or large numbers of files. It also reduces the load on your backend applications and servers, and it reduces the number of network hops. A [shared access signature \(SAS\)](#) (SAS) enables you to securely provide your client applications with access to objects in Azure Storage.

Shared access signatures can be used to restrict the scope of operations that a client can perform, and the objects that they can perform operations against. For example, if you have a shared storage account for all of your tenants, and you store all of tenant A's data in a blob container named `tenantA`, you can create an SAS that only permits tenant A's users to access that container. For more information, see [Isolation models](#) to explore the approaches you can use to isolate your tenants' data in a storage account.

The [Valet Key pattern](#) is useful as a way to issue constrained and scoped shared access signatures from your application tier. For example, suppose you have a multitenant application that allows users to upload videos. Your API or application tier can authenticate the client using your own authentication system. You can then provide a

SAS to the client that allows them to upload a video file to a specified blob, into a container and blob path that you specify. The client then uploads the file directly to the storage account, avoiding the extra bandwidth and load on your API. If they try to read data from the blob container, or if they try to write data to a different part of the container to another container in the storage account, Azure Storage blocks the request. The signature expires after a configurable time period.

[Stored access policies](#) extend the SAS functionality, which enables you to define a single policy that can be used when issuing multiple shared access signatures.

Identity-based access control

Azure Storage also provides [identity-based access control](#) by using Azure Active Directory (Azure AD). This capability also enables you to use [attribute-based access control](#), which gives you finer-grained access to blob paths, or to blobs that have been tagged with a specific tenant ID.

Lifecycle management

When you use blob storage in a multitenant solution, your tenants might require different policies for data retention. When you store large volumes of data, you might also want to configure the data for a specific tenant to automatically be moved to the [cool or archive storage tiers](#), for cost-optimization purposes.

Consider using [lifecycle management policies](#) to set the blob lifecycle for all tenants, or for a subset of tenants. A lifecycle management policy can be applied to blob containers, or to a subset of blobs within a container. However, there are limits on the number of rules you can specify in a lifecycle management policy. Make sure you plan and test your use of this feature in a multitenant environment, and consider deploying multiple storage accounts, if you will exceed the limits.

Immutable storage

When you configure [immutable blob storage](#) on storage containers with [time-based retention policies](#), Azure Storage prevents deletion or modification of the data before a specified time. The prevention is enforced at the storage account layer and applies to all users. Even your organization's administrators can't delete immutable data.

Immutable storage can be useful when you work with tenants that have legal or compliance requirements to maintain data or records. However, you should consider how this feature is used within the context of your [tenant lifecycle](#). For example, if

tenants are offboarded and request the deletion of their data, you might not be able to fulfill their requests. If you use immutable storage for your tenants' data, consider how you address this issue in your terms of service.

Server-side copy

In a multitenant system, there is sometimes a need to move data from one storage account to another. For example, if you move a tenant between deployment stamps or rebalance a [sharded](#) set of storage accounts, you need to copy or move a specific tenant's data. When working with large volumes of data, it's advisable to use [server-side copy APIs](#) ↗ to decrease the time it takes to migrate the data.

The [AzCopy tool](#) is an application that you can run from your own computer, or from a virtual machine, to manage the copy process. AzCopy is compatible with the server-side copy feature, and it provides a scriptable command-line interface that you can run from your own solutions. AzCopy is also helpful for uploading and downloading large volumes of data.

If you need to use the server-side copy APIs directly from your code, consider using the [Put Block From URL API](#), [Put Page From URL API](#), [Append Block From URL API](#), and the [Copy Blob From URL API](#) when working with smaller blobs.

Object replication

The [Object replication](#) feature automatically replicates data between a source and destination storage account. Object replication is asynchronous. In a multitenant solution, this feature can be useful when you need to continuously replicate data between deployment stamps, or in an implementation of the [Geode pattern](#).

Encryption

Azure Storage enables you to [provide encryption keys](#) for your data. In a multitenant solution, consider combining this capability with [encryption scopes](#), which enable you to define different encryption keys for different tenants, even if their data is stored in the same storage account. By using these features together, you can also provide tenants with control over their own data. If they need to deactivate their account, they can delete the encryption key and their data is no longer accessible.

Monitoring

When working with a multitenant solution, consider whether you need to [measure the consumption for each tenant](#), and define the specific metrics you need to track, such as the amount of storage used for each tenant (the capacity), or the number of operations performed for each tenant's data. You can also use [cost allocation](#) to track the cost of each tenant's usage and enable chargeback across multiple subscriptions.

Azure Storage provides [built-in monitoring capabilities](#). It's important to consider the services you'll use within the Azure Storage account. For example, when you work with [blobs](#), it's possible to view the total capacity of a storage account, but not a single container. In contrast, when you work with file shares, it's possible to see the capacity for each share, but not for each folder.

You can also [log all of the requests made to Azure Storage](#), and then you can aggregate and analyze those logs. This provides more flexibility in how you aggregate and group data for each tenant. However, in solutions that create high volumes of requests to Azure Storage, it's important to consider whether the benefit you gain from this approach justifies the cost involved in capturing and processing those logs.

[Azure Storage inventory](#) provides another approach to measure the total size of a blob container.

Isolation models

When working with a multitenant system using Azure Storage, you need to make a decision about the level of isolation you want to use. Azure Storage supports several isolation models.

Storage accounts per tenant

The strongest level of isolation is to deploy a dedicated storage account for a tenant. This ensures that all storage keys are isolated and can be rotated independently. This approach enables you to scale your solution to avoid limits and quotas that are applicable to each storage account, but you also need to consider the maximum number of storage accounts that can be deployed into a single Azure subscription.

Note

Azure Storage has many quotas and limits that you should consider when you select an isolation model. These include [Azure service limits](#), [scalability targets](#), and [scalability targets for the Azure Storage resource provider](#).

Additionally, each component of Azure Storage provides further options for tenant isolation.

Blob storage isolation models

The following table summarizes the differences between the main tenancy isolation models for Azure Storage blobs:

Consideration	Shared blob containers	Blob containers per tenant	Storage accounts per tenant
Data isolation	Low-medium. Use paths to identify each tenant's data, or hierarchical namespaces	Medium. Use container-scoped SAS URLs to support security isolation	High
Performance isolation	Low	Low. Most quotas and limits apply to entire storage account	High
Deployment complexity	Low	Medium	High
Operational complexity	Low	Medium	High
Example scenario	Storing a small number of blobs per tenant	Issue tenant-scoped SAS URLs	Separate deployment stamps for each tenant

Shared blob containers

When working with blob storage, you might choose to use a shared blob container, and you might then use blob paths to separate data for each tenant:

Tenant ID	Example blob path
tenant-a	https://contoso.blob.core.windows.net/sharedcontainer/tenant-a/blob1.mp4
tenant-b	https://contoso.blob.core.windows.net/sharedcontainer/tenant-b/blob2.mp4

While this approach is simple to implement, in many scenarios, blob paths don't provide sufficient isolation across tenants. This is because blob storage doesn't typically provide a concept of directories or folders. This means you can't assign access to all blobs within a specified path. However, Azure Storage provides a capability to [list \(enumerate\) blobs](#)

that begin with a specified prefix, which can be helpful when you work with shared blob containers and don't require directory-level access control.

Azure Storage's [hierarchical namespace](#) feature provides the ability to have a stronger concept of a directory or folder, including directory-specific access control. This can be useful in some multitenant scenarios where you have shared blob containers, but you want to grant access to a single tenant's data.

In some multitenant solutions, you might only need to store a single blob or set of blobs for each tenant, such as tenant icons for customizing a user interface. In these scenarios, a single shared blob container might be sufficient. You could use the tenant identifier as the blob name, and then read a specific blob instead of enumerating a blob path.

When you work with shared containers, consider whether you need to track the data and Azure Storage service usage for each tenant, and plan an approach to do so. See [Monitoring](#) for further information.

Blob containers per tenant

You can create individual blob containers for each tenant within a single storage account. There is no limit to the number of blob containers that you can create, within a storage account.

By creating containers for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data. You can also easily monitor the capacity that each container uses.

File storage isolation models

The following table summarizes the differences between the main tenancy isolation models for Azure Storage files:

Consideration	Shared file shares	File shares per tenant	Storage accounts per tenant
Data isolation	Medium-high. Apply authorization rules for tenant-specific files and directories	Medium-high	High
Performance isolation	Low	Low-medium. Most quotas and limits apply to the entire storage account, but set size quotas on a per-share level	High

Consideration	Shared file shares	File shares per tenant	Storage accounts per tenant
Deployment complexity	Low	Medium	High
Operational complexity	Low	Medium	High
Example scenario	Application controls all access to files	Tenants access their own files	Separate deployment stamps for each tenant

Shared file shares

When working with file shares, you might choose to use a shared file share, and then you might use file paths to separate data for each tenant:

Tenant ID	Example file path
tenant-a	https://contoso.file.core.windows.net/share/tenant-a/blob1.mp4
tenant-b	https://contoso.file.core.windows.net/share/tenant-b/blob2.mp4

When you use an application that can communicate using the Server Message Block (SMB) protocol, and when you use Active Directory Domain Services either on-premises or in Azure, file shares [support authorization](#) at both the share and the directory/file levels.

In other scenarios, consider using SAS to grant access to specific file shares or files. When you use SAS, you can't grant access to directories.

When you work with shared file shares, consider whether you need to track the data and Azure Storage service usage for each tenant, and then plan an approach to do so (as necessary). See [Monitoring](#) for further information.

File shares per tenant

You can create individual file shares for each tenant, within a single storage account. There is no limit to the number of file shares that you can create within a storage account.

By creating file shares for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data. You can also easily monitor the

capacity each file share uses.

Table storage isolation models

The following table summarizes the differences between the main tenancy isolation models for Azure Storage tables:

Consideration	Shared tables with partition keys per tenant	Tables per tenant	Storage accounts per tenant
Data isolation	Low. Application enforces isolation	Low-medium	High
Performance isolation	Low	Low. Most quotas and limits apply to entire storage account	High
Deployment complexity	Low	Medium	High
Operational complexity	Low	Medium	High
Example scenario	Large multitenant solution with shared application tier	Issue tenant-scoped SAS URLs	Separate deployment stamps for each tenant

Shared tables with partition keys per tenant

When using table storage with a single shared table, you can consider using the [built-in support for partitioning](#). Each entity must include a partition key. A tenant identifier is often a good choice for a partition key.

Shared access signatures and policies enable you to specify a partition key range, and Azure Storage ensures that requests containing the signature can only access the specified partition key ranges. This enables you to implement the [Valet Key pattern](#), which allows untrusted clients to access a single tenant's partition, without affecting other tenants.

For high-scale applications, consider the maximum throughput of each table partition and the storage account.

Tables per tenant

You can create individual tables for each tenant within a single storage account. There is no limit to the number of tables that you can create within a storage account.

By creating tables for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data.

Queue storage isolation models

The following table summarizes the differences between the main tenancy isolation models for Azure Storage queues:

Consideration	Shared queues	Queues per tenant	Storage accounts per tenant
Data isolation	Low	Low-medium	High
Performance isolation	Low	Low. Most quotas and limits apply to entire storage account	High
Deployment complexity	Low	Medium	High
Operational complexity	Low	Medium	High
Example scenario	Large multitenant solution with shared application tier	Issue tenant-scoped SAS URLs	Separate deployment stamps for each tenant

Shared queues

If you choose to share a queue, consider the quotas and limits that apply. In solutions with a high request volume, consider whether the target throughput of 2,000 messages per second is sufficient.

Queues don't provide partitioning or subqueues, so data for all tenants could be intermingled.

Queues per tenant

You can create individual queues for each tenant within a single storage account. There is no limit to the number of queues that you can create within a storage account.

By creating queues for each tenant, you can use Azure Storage access control, including SAS, to manage access for each tenant's data.

When you dynamically create queues for each tenant, consider how your application tier will consume the messages from each tenant's queue. For more advanced scenarios, consider using [Azure Service Bus](#), which supports features such as [topics and subscriptions](#), [sessions](#), and [message auto-forwarding](#), which can be useful in a multitenant solution.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Dr. Christian Geuer-Pollmann](#) | Principal Customer Engineer, FastTrack for Azure
- [Patrick Horn](#) | Senior Customer Engineering Manager, FastTrack for Azure
- [Ben Hummerstone](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [Vic Perdana](#) | Cloud Solution Architect, Azure ISV

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [storage and data approaches for multitenancy](#).

Multitenancy and Azure Event Hubs

Article • 10/10/2023

[Event Hubs](#) is a big data streaming platform and event ingestion service that can receive and process millions of events per second. You can transform and store event hub data by using real-time analytics providers and batching/storage adapters. For a comparison of Event Hubs and other Azure messaging services, see [Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus](#).

This article describes Event Hubs features and isolation models that you can use in multitenant solutions.

Isolation models

When you use Event Hubs in your multitenant system, you need to decide the level of isolation that you want. Event Hubs supports different models of multitenancy.

- **Trusted multitenancy:** All tenants share an Event Hubs namespace. This choice can be appropriate when all the tenants are in your organization.
- **Hostile multitenancy:** Each tenant has its own namespace that isn't shared. This choice can be appropriate when you want to ensure that your tenants don't have [noisy neighbor](#) problems.

A system can implement both models: some tenants share namespaces, others have a dedicated one. Also, a tenant can share a namespace with other tenants but have dedicated event hubs.

The following table summarizes the differences between the main tenancy isolation models for Event Hubs. The models are described in more detail in the sections that follow.

Consideration	Dedicated namespace	Shared namespace, dedicated event hubs	Shared namespace and event hubs
Data isolation	High	Medium	None
Performance isolation	Highest. Manage performance needs based on each tenant's requirements.	Medium. Can have noisy neighbor issues.	Low. Can have noisy neighbor issues.

Consideration	Dedicated namespace	Shared namespace, dedicated event hubs	Shared namespace and event hubs
Deployment complexity	Medium. Be aware of Event Hubs quotas and limits at the subscription level.	Medium. Separate message entities must be deployed for each tenant. Be aware of Event Hubs quotas and limits . Some cases require multiple namespaces, depending on the number of tenants.	Low
Operational complexity	High. Need to manage namespaces on a per-tenant basis.	Medium. Some tenants require granular management of message entities.	Low
Example scenario	Separate application instances per tenant.	Dedicated event hubs for each tenant.	Large multitenant solution with a shared application tier and one or more shared event hubs.

ⓘ Note

Event Hubs for Apache Kafka is a feature that provides a protocol head on top of Event Hubs so that Event Hubs can be used by Apache Kafka applications. The applications stream events into event hubs, which are equivalent to Kafka topics. For more information, see [What is Azure Event Hubs for Apache Kafka](#).

Dedicated namespace

In this model, you provision an [Event Hubs namespace](#) for each tenant. This approach provides the maximum level of isolation and the ability to provide acceptable performance for all tenants.

You can use the following techniques to fine-tune eventing capabilities to satisfy tenant requirements:

- Deploy the namespace to a region that's close to the tenant.
- Deploy the namespace with a pricing tier that's appropriate to the tenant. For example, if you use a [premium namespace](#) you can choose the number of [processing units](#).

- Apply networking restrictions that are based on tenant needs by using [IP firewall rules](#), [private endpoints](#), and [virtual network service endpoints](#).
- Use [tenant-specific encryption keys](#).
- Configure [Event Hubs Geo-disaster recovery](#) and [availability zones](#) to meet tenant availability requirements.

If you reach the maximum number of Event Hubs namespaces in your Azure subscription, you can deploy namespaces across different subscriptions by using the [Deployment Stamps pattern](#).

The disadvantage of this isolation model is that, as the number of tenants grows over time, managing the namespaces gets more complex. Another disadvantage is that the model increases costs, because you pay for each namespace.

Shared namespace, dedicated event hubs

Even if a namespace is shared by multiple tenants, you can isolate tenants to a dedicated event hub. You can use shared access signatures or Microsoft Entra identities to control access.

As the number of tenants grows within your system, the number of event hubs also increases to accommodate each tenant. This growth can lead to higher operational costs and lower organizational agility. There's a [limit](#) on the number of event hubs per namespace. Thus the number of namespaces that your system requires depends on the number of event hubs that your tenants require.

When a namespace is shared, [noisy neighbor](#) problems are more likely. For example, it's possible that the event entities of a tenant could consume a disproportionate amount of the namespace resources and hinder other tenants. Event hub namespaces have limits on their processing units (premium tier) or capacity units (dedicated tier) and on the number of brokered connections to a namespace. Consider whether a single tenant might consume too many resources.

Shared namespace and event hubs

You can have a namespace and event entities that are shared by all your tenants. This model decreases operational complexity and lowers resource costs.

However, having a shared namespace can lead to the [noisy neighbor](#) problem and result in higher latency for some tenants. You also have to implement your applications to serve multiple tenants. Shared event hubs and Kafka topics don't provide data isolation

between tenants, so you have to satisfy data isolation requirements in your application logic.

ⓘ Note

Don't use Event Hubs partitions to try to isolate your tenants. Partitioning in Event Hubs enables the processing of events and scalability, but it isn't an isolation model. You can send events directly to partitions, but doing so isn't recommended because it downgrades the availability of an event hub to partition level. For more information, see [Availability and consistency in Event Hubs](#).

Features of Event Hubs that support multitenancy

The following features of Event Hubs support multitenancy:

- [Application groups](#)
- [Microsoft Entra authentication](#)
- [Shared access signature](#)
- [Customer-managed keys](#)
- [Event Hubs Capture](#)
- [Geo-disaster recovery](#)
- [IP firewall rules](#)

These features are discussed in the following sections.

Application groups

An application group is a collection of one or more client applications that interact with the Event Hubs data plane. You can apply quota and access management policies to all the applications in the group by applying them to the group itself.

Each application group can be scoped to a single Event Hubs namespace or to a single event hub. It should use a uniquely identifying condition identifier of the client applications, such as the security context, which is either a shared access signature (SAS) or a Microsoft Entra application ID.

For more information, see [Resource governance with application groups](#).

Microsoft Entra authentication

Event Hubs is integrated with Microsoft Entra ID. Clients can authenticate to Event Hubs resources by using a managed identity with Microsoft Entra ID. Event Hubs defines a set of built-in roles that you can grant to your tenants to access Event Hubs entities. For example, by using Microsoft Entra authentication, you can grant a tenant access to an event hub that has the messages for that tenant. You can use this technique to isolate a tenant from other tenants.

Kafka applications can use [managed identity OAuth](#) to access Event Hubs resources.

For more information, see the following articles:

- [Authenticate a managed identity with Microsoft Entra ID to access Event Hubs resources](#)
- [Authenticate an application with Microsoft Entra ID to access Event Hubs resources](#)

Shared access signature

Shared access signatures (SAS) give you the ability to grant a tenant access to Event Hubs resources with specific rights. If you isolate your tenants at an event entity level, you can grant SAS keys on an event hub or Kafka topic that applies only to a particular tenant.

For more information, see [Authenticate access to Event Hubs resources using shared access signatures \(SAS\)](#)

Customer-managed keys

If your tenants require their own keys to encrypt and decrypt events, you can configure customer-managed keys in some versions of Event Hubs.

This feature requires that you use the [dedicated namespace](#) isolation model. Encryption can only be enabled for new or empty namespaces.

For more information, see [Configure customer-managed keys for encrypting Azure Event Hubs data at rest](#).

Event Hubs Capture

You can use the Event Hubs Capture feature to automatically capture streaming data from Event Hubs and store it to an Azure Blob Storage or Data Lake Storage account.

This capability is useful for archiving events. For example, if you're required to archive events for a tenant for compliance reasons, you can deploy tenant-specific namespaces

and enable Event Hubs Capture to archive events to tenant-specific Azure Storage Accounts. You can also enable Event Hubs Capture on tenant-specific event hubs in a shared namespace.

For more information, see [Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage](#)

Geo-disaster recovery

Geo-disaster recovery continuously replicates the entire configuration of an Event Hubs namespace from a primary namespace to a secondary namespace that's paired with the primary. This feature can help to recover from disasters, such as regional failures.

For example, if you isolate your tenants at the namespace level, you can replicate the configuration of a tenant namespace to a secondary region to provide protection against outages and failure of the primary.

For more information, see [Azure Event Hubs - Geo-disaster recovery](#).

Note

To help protect continuity of operations, Geo-disaster recovery replicates the configuration of the primary namespace to the secondary namespace. It doesn't replicate the event data, nor does it replicate any Microsoft Entra RBAC assignments that you use for your primary namespace. For more information, see [Multi-site and multi-region federation](#).

IP firewall rules

You can use IP firewall rules to control access to namespaces. When you isolate tenants at the namespace level, you can configure the namespaces to accept connections only from clients that originate from allowed IP addresses or address ranges.

For more information, see:

- [Network security for Azure Event Hubs](#)
- [Allow access to Azure Event Hubs namespaces from specific IP addresses or ranges](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- Will Velida | Customer Engineer 2, FastTrack for Azure

Other contributors:

- John Downs | Principal Customer Engineer, FastTrack for Azure
- Paolo Salvatori | Principal Customer Engineer, FastTrack for Azure
- Arsen Vladimirskiy | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Azure Event Hubs — A big data streaming platform and event ingestion service](#)
- [What is Azure Event Hubs for Apache Kafka](#)
- Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage
- [Overview of Event Hubs Premium](#)
- [Overview of Azure Event Hubs Dedicated tier](#)
- [Event Hubs documentation](#)
- [Learn: Enable reliable messaging for Big Data applications using Azure Event Hubs](#)

Related resources

- [Event-driven architecture style](#)
- [Architectural approaches for messaging in multitenant solutions](#)
- [Messaging patterns](#)
- [Serverless event processing](#)
- [Integrate Event Hubs with serverless functions on Azure](#)
- [Monitor Azure Functions and Event Hubs](#)
- [Performance and scale for Event Hubs and Azure Functions](#)
- [Partitioning in Azure Event Hubs and Kafka](#)

Multitenancy and Azure Service Bus

Article • 10/10/2023

Azure Service Bus provides highly reliable and asynchronous cloud messaging between applications and services that aren't necessarily online at the same time. Service Bus is a fully managed enterprise-level message broker with support for both message queues and publish-subscribe topics. In this article, we describe some of the features of Service Bus that are useful for multitenant solutions. We then provide links to the guidance that can help you when you're planning how you're going to use Service Bus.

Isolation models

When working with a multitenant system that uses Service Bus, you need to make a decision about the level of isolation that you want to adopt. Service Bus supports several isolation models.

The following table summarizes the differences between the main tenancy models for Service Bus:

Consideration	Namespace per tenant	Shared namespace, separate topics or queues per tenant	Shared namespace, topics or queues between tenants
Data isolation	High	Medium	None
Performance isolation	Highest. Manage performance needs based on each tenant's requirements	Medium. Potentially subject to noisy neighbor issues	Low. Potentially subject to noisy neighbor issues
Deployment complexity	Medium. Be aware of Azure Service Bus quotas and limits at the subscription level	Medium. Message entities must be deployed on a per-tenant basis. Be aware of Azure Service Bus quotas and limits at the namespace level	Low
Operational complexity	High. Need to manage namespaces on a per-tenant basis	Medium. Granular management of message entities might be required depending on tenant	Low
Example scenario	Individual application instances per tenant	Dedicated queues for each tenant	Large multitenant solution with a shared application tier and

Consideration	Namespace per tenant	Shared namespace, separate topics or queues per tenant	Shared namespace, topics or queues between tenants
			one or more shared queues and topics

Dedicated namespace per tenant

Within your solution, you can use a specific Service Bus namespace for each tenant. This deployment approach provides your solution with the maximum level of isolation, with the ability to provide consistent performance per tenant.

You can also fine-tune messaging capabilities for each tenant based on their needs, such as by using the following approaches:

- Deploy the namespace to a region that's close to the tenant.
- Deploy a tenant-specific namespace with a pricing tier that's appropriate to that tenant. For example, you can provision [premium namespaces](#) with a different number of [messaging units](#).
- Apply networking restrictions based on the tenant's needs.
- Use [tenant-specific encryption keys](#).

The disadvantage to this isolation model is that, as the number of tenants grows within your system over time, the operational complexity of managing your namespaces also increases. If you reach the maximum number of namespaces per Azure subscription, you could deploy namespaces across different subscriptions (see [deployment stamp pattern](#)). This approach also increases resource costs, since you pay for each namespace you provision.

Separate topics and queues in a shared namespace

You can isolate your tenants on a messaging entity level. For example, each tenant within your system can have a dedicated one or more queues that it listens to. You can authenticate and authorize access to each tenant's messaging entity with a different shared access signature or Microsoft Entra identity.

As the number of tenants grows within your system, the number of queues, topics, or subscriptions also increases to accommodate each tenant. This growth might lead to higher operational costs and lower organizational agility.

Because the namespace is shared across all tenants, [noisy neighbor](#) problems are more likely with this approach. For example, it's possible that a single tenant's messaging

entities could consume a disproportionate amount of the namespace resources and impact all the other tenants. Service Bus namespaces have limits around the messaging unit capacity and the number of concurrent connections to a namespace. Consider whether a single tenant might consume more than their fair share of these resources.

There are also [limits](#) on how many topics and queues you can provision within a single namespace, although these limits are higher than the limit of namespaces in a subscription.

Shared topics or queues

Use the same namespace and messaging entities for all your tenants to decrease your operational complexity and to lower your resource costs. It can also unlock advanced messaging and [filtering](#) scenarios.

However, having a single namespace that all your tenants share can also lead to the [noisy neighbor](#) problem, which might cause higher latency for some tenants. You'll also need to make your application code multitenancy-aware. For example, you can pass the tenant context within the message payload or in a user-defined property. This approach enables you to correctly handle messages that are intended for different tenants. Service Bus provides topic filters and actions, which can be used to define which message a particular subscriber should receive. When using a shared topic or queue, there's no data isolation between tenants, so you'll need to implement your data isolation requirements within your application logic.

Note

Sessions are a useful feature to support message ordering requirements. However, they aren't typically used for isolating data between tenants, unless you also have requirements for message ordering within a tenant. For most multitenant solutions, you should consider using one of the other isolation models described in this article.

Features of Azure Service Bus that support multitenancy

Microsoft Entra authentication

Service Bus is integrated with Microsoft Entra ID, which allows clients to authenticate a managed identity with Microsoft Entra ID to Service Bus resources. Service Bus defines a set of built-in roles that you can grant to your tenants to access Service Bus entities. For example, with Microsoft Entra authentication, you can grant a tenant access to a specific queue or topic that contains their messages, which isolates it from the other tenants within your application.

See [Authenticate a managed identity with Microsoft Entra ID to access Azure Service Bus resources](#).

Customer-managed keys

If your tenants need to use their own keys to encrypt and decrypt messages, you can configure customer-managed keys in Service Bus premium namespaces. This feature requires that you adopt the [dedicated namespace-per-tenant](#) isolation model.

See [Configure customer-managed keys for encrypting Azure Service Bus data at rest](#).

Shared access signatures

Shared access signatures (SAS) give you the ability to grant a tenant access to Service Bus resources with specific rights. If you choose to isolate your tenants at a messaging entity level, you can grant SAS keys on a queue or topic that only apply to a particular tenant.

For more information, see the following articles:

- [Shared Access Signature in Azure Service Bus](#)
- [Service Bus access control with Shared Access Signatures](#)

Topic filters and actions

If you're using Service Bus topics within your namespace, you can use topic filters and actions to allow your subscribers to define which messages they want to receive from a topic. Each rule has a filter condition that selects the desired message, along with an action that will annotate the message. For example, if your topic subscriptions are split on a per-tenant basis, you can use filters to receive messages from a topic that are only intended for that tenant.

See [Topic filters and actions](#).

Suspend and reactivate messaging entities

You can temporarily suspend message entities. Suspension puts the messaging entity into a disabled state, and all messages are maintained in storage. The ability to deactivate messaging entities is useful when handling your [tenant lifecycle](#). For example, if a tenant unsubscribes from your product, you could disable the queues that are specific to that particular tenant.

See [Suspend and reactivate messaging entities \(disable\)](#).

Partitioning

Partitions can be used to improve the throughput of your messaging entities by distributing the messages across multiple message brokers and message stores.

You can assign a partition to a specific tenant by setting the message's partition key to that tenant's identifier. This approach ensures that Service Bus assigns the message to that tenant's partition. However, Service Bus has limits on the number of partitions that you can support on a single entity. For shared entities, you should consider using partition keys that are derived from the tenant ID. For example, you could use a hashing algorithm that converts tenant IDs to a fixed number of partition keys.

Partitioning is available when you deploy namespaces with specific SKUs. For more information, see [Service Bus Premium and Standard messaging tiers](#).

See [Partitioned queues and topics](#).

ⓘ Note

Partitioning is available at entity creation for all queues and topics in Basic or Standard SKUs. It isn't available for the Premium messaging SKU anymore, but any previously existing partitioned entities from when they were supported in Premium namespaces will continue to work as expected.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Will Velida](#) | Customer Engineer 2, FastTrack for Azure

Other contributors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Daniel Larsen](#) | Principal Customer Engineer, FastTrack for Azure
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [architectural approaches for messaging in multitenant solutions](#)

Multitenancy and Application Insights

Article • 08/21/2023

Application Insights is a service that monitors the performance, availability, and usage of your web applications. It can help you identify and diagnose problems, analyze user behavior, and track key metrics. This article describes some of the features of Application Insights that are useful for multitenant systems. It also describes various tenancy models.

💡 Tip

Application Insights is designed and optimized for monitoring solutions. It's not intended to be used to capture every event that happens in a system, as you might need to do for auditing or billing. To learn about how you can measure usage for billing purposes, see [Considerations for measuring consumption in multitenant solutions](#).

Isolation models

When you implement a multitenant system that uses Application Insights, you need to determine the required level of isolation. There are several isolation models that you can choose from. Here are some factors that might influence your choice:

- How many tenants do you plan to have?
- Do you share your application tier among multiple tenants, or do you deploy separate deployment stamps for each tenant?
- Are you or your customers sensitive about storing data alongside other tenants' data?
- Is the application tier of your solution multitenant, and the data tier single tenant?
- Do telemetry requirements vary among tenants?

💡 Tip

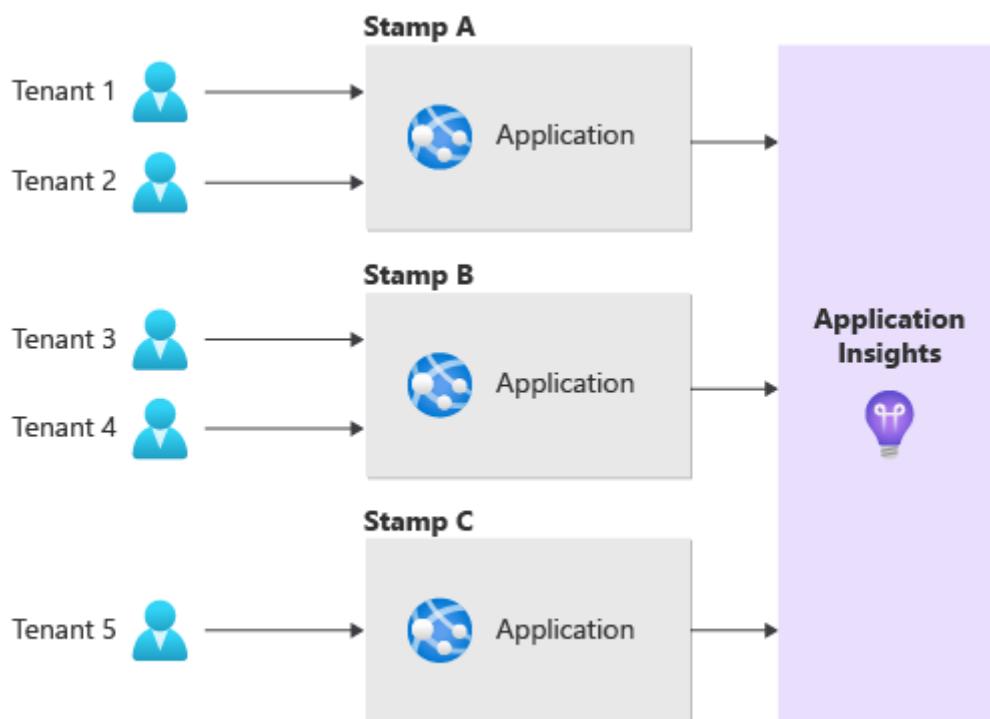
The main factors that determine the cost of Application Insights are the amount of data that you send to it and how long the data is retained. In a multitenant application, the overall cost is the same for a dedicated Application Insights instance as it is for a shared instance. For more information, see the [Azure Monitor pricing page](#).

This table summarizes the differences between the main tenancy models for Application Insights:

Consideration	Globally shared Application Insights instance	One Application Insights instance per region/stamp	One Application Insights instance per tenant
Data isolation	Low	Low	High
Performance isolation	Low	Medium	High
Deployment complexity	Low to medium, depending on the number of tenants	Medium, depending on the number of tenants	High
Operational complexity	Low	Medium	High
Example scenario	Large multitenant solution with a shared application tier	Multitenant solution with regional deployments to better serve a global customer base	Individual application instances per tenant

Globally shared Application Insights instance

You can use a single instance of Application Insights to track telemetry for tenants in a multitenant application, as shown here:



Benefits of this approach include simplified configuration and management of the application, because you only need to instrument the application code once. Drawbacks of this approach include the limits and quotas that are associated with a single Application Insights instance. To determine whether limits might affect your multitenant application, see the [Application Insights limits](#).

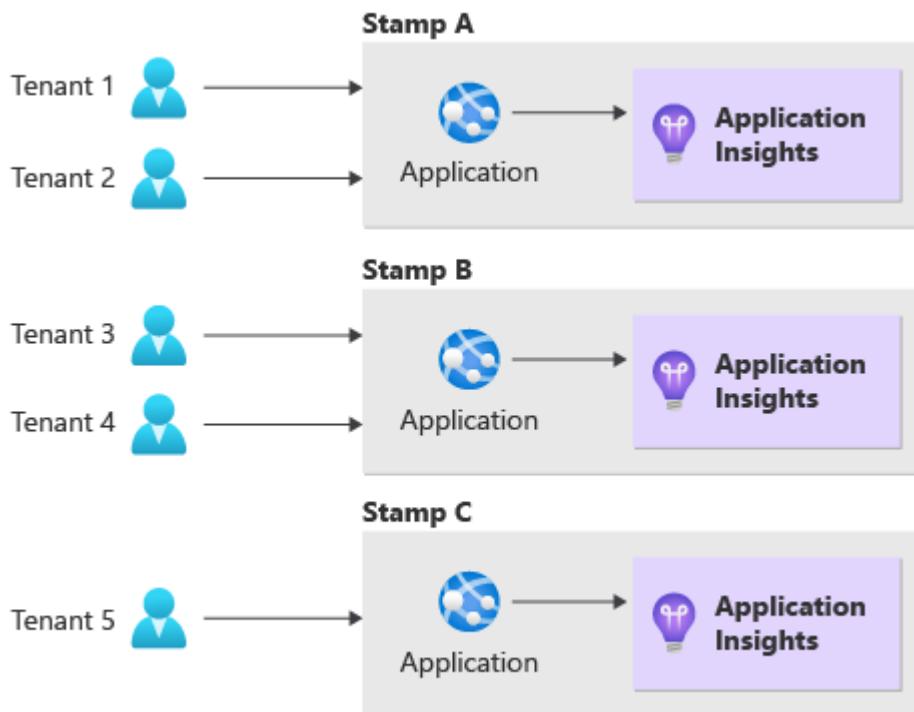
When you use a shared Application Insights resource, it might also be more difficult to isolate and filter the data for each tenant, especially if you have a large number of tenants. Because all tenants share the same Log Analytics workspace and instrumentation keys, security and privacy might also be a concern.

To address these concerns, you might need to implement logic and mechanisms to ensure that data can be filtered by tenant and that your operations team can properly see per-tenant data. You can implement filtering by adding a [custom property](#) to capture the tenant ID as part of every telemetry item. The tenant ID can then be used to query the data.

One Application Insights instance per stamp

Multitenant solutions often include multiple stamps, which might be deployed in different Azure regions. Stamps enable you to serve tenants that are local to a particular region so you can provide better performance. A single stamp might serve a single tenant or a subset of your tenants. To learn more about stamps, see [Deployment stamps pattern](#).

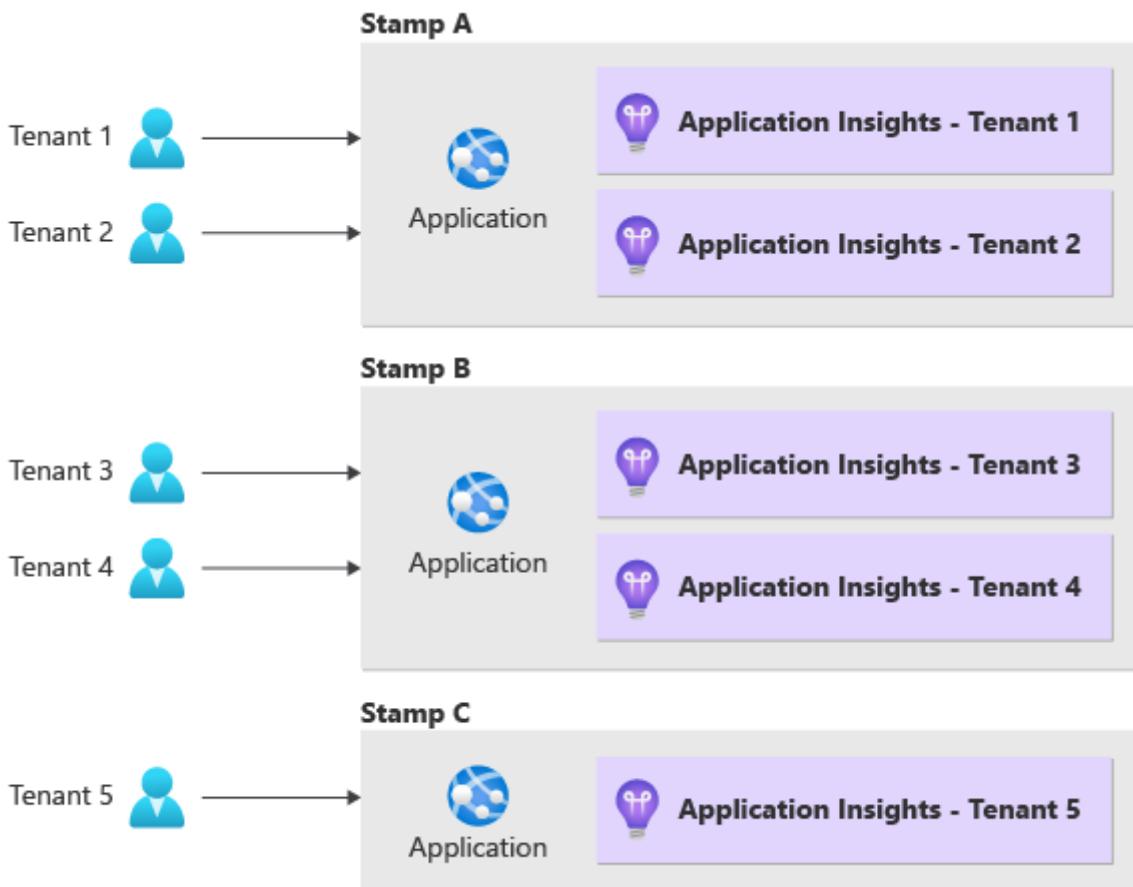
You might decide to deploy an Application Insights instance in each stamp, sharing the instance among all tenants that use the stamp, as shown here:



This approach provides more flexibility with resource limits because the limits apply per instance of Application Insights.

One Application Insights instance per tenant

You might decide to use a dedicated Application Insights instance for each tenant:



This approach gives you more flexibility and control over tenants' telemetry data and provides the strongest data isolation. When you use this model, you can configure tenant-specific settings and retention policies.

When you use this approach, however, you need to deploy a large number of Application Insights instances, manage-tenant specific settings in a tenant catalog, and change application code when new tenants are onboarded. Note that the decision to deploy a dedicated Application Insights instance per tenant is separate from the decision to deploy an application tier for each tenant. For example, you can decide to deploy a single application instance in a stamp that's shared by multiple tenants but deploy one Application Insights instance for each tenant.

You should consider using one Application Insights instance per tenant if you require a high degree of data isolation between your tenants, you need different configurations for various tenants, or the service limits of a single Application Insights instance don't meet your needs.

With this approach, it might be difficult to aggregate data and compare it across all tenants because you need to query multiple Application Insights instances separately. If you use this approach, consider using [cross-resource queries and Azure Monitor workbooks](#).

Application Insights features that support multitenancy

Custom properties and metrics

Application Insights provides a way to enrich telemetry data with custom properties and metrics. *Custom properties* are key-value pairs that you can attach to any telemetry item, such as requests or events. *Custom metrics* are numerical values that you can track over time, like a score or the length of a queue. You can use custom properties and metrics to add tenant-specific information, like tenant ID, tenant name, tenant location, and deployment stamp ID, to telemetry data.

There are two ways to add custom properties to your telemetry: by using `TelemetryClient` or by using telemetry initializers.

TelemetryClient

[TelemetryClient](#) is an object that you can use to track any telemetry item. You can access the custom properties of any telemetry item via its `Properties` dictionary. The advantage of using `TelemetryClient` is that you have full control over what custom properties to add, and when to add them. The disadvantage is that you need to access and modify each telemetry item that you want to enrich with custom properties.

Telemetry initializers

You can use [telemetry initializers](#) to add information to all telemetry items, or to modify properties that are set by the standard telemetry modules.

When you share an Application Insights instance across multiple tenants, a telemetry initializer often provides a good way to inject the tenant ID into every telemetry item. You can then use the ID to query and filter for reporting. The advantage of using telemetry initializers is that you can apply custom properties to all or some of the telemetry items in one place without needing to write code for each item. The disadvantage is that you have less control over which custom properties to add to each telemetry item, so you might add unnecessary or redundant data.

When you add custom properties to telemetry data, by using either mechanism, you can use powerful features of Application Insights to monitor and analyze multitenant applications in a more granular and meaningful way. For example, you can:

- Use metrics explorer to create charts and graphs that show the performance and usage of the application for each tenant.
- Use Log Analytics to write complex queries that filter, aggregate, and join telemetry data based on tenant-specific properties or metrics.
- Use alerts to set up rules that notify you when certain conditions are met for a tenant.
- Use Azure Monitor workbooks to create interactive reports and dashboards that visualize the health and status of the application for each tenant.

Unify multiple Application Insights instances into a single view

There are several ways to unify data from multiple Application Insights instances. Your choice depends on your needs and preferences. The following sections describe some of the options.

Cross-resource queries

You can use cross-resource queries to [query](#) data from multiple Application Insights instances in a single query. The instances can be in a single resource group, in more than one resource group, or in more than one subscription. As the number of Application Insights workspaces in a query increases, query performance might degrade. The number of Application Insights workspaces that you can include in a single query is also limited. For more information, see [Query across multiple workspaces and apps](#).

Azure Monitor workbooks

You can use [Azure Monitor workbooks](#) to create interactive reports and dashboards based on data from multiple sources, including Application Insights. These reports and dashboards enable you to visualize and analyze data from multiple Application Insights instances in a single view.

Latency

The time it takes for data on a monitored system to become available for analysis is referred to as [latency](#). A shared Application Insights instance in a multitenant application doesn't incur more latency than a dedicated one unless the shared instance is throttled and the throttling prevents data from being ingested. In that scenario, latency increases.

Rate limiting on ingestion

You can perform ingestion rate limiting in Application Insights by using [sampling](#) to limit the amount of telemetry data that's ingested by your service per day. Sampling helps prevent Application Insights from throttling telemetry due to ingestion limits. You can use fixed-rate sampling to determine an optimal sampling rate, based on the number of tenants and the daily cap, in order to stay within the limits.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Raj Nemani](#) | Director, Partner Technology Strategist, GPS-ISV

Other contributors:

- [Mick Alberts](#) | Technical Writer
- [Rob Bagby](#) | Principal Content Developer, C+E Skilling Content R&D
- [John Downs](#) | Principal Program Manager, MCAPS
- [Rick Hallihan](#) | Senior Software Engineer, C+E Skilling Content R&D
- [Landon Pierce](#) | Customer Engineer, Azure CXP
- [Daniel Scott-Raynsford](#) | Partner Technology Strategist, OCP
- [Arsen Vladimirsksiy](#) | Principal Customer Engineer, Azure CXP

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Training: Monitor app performance](#)
- [What is Application Insights?](#)
- [Application Insights limits](#)
- [Query across multiple workspaces and apps](#)
- [Training: Visualize data combined from multiple data sources by using Azure Workbooks](#)
- [Capture Application Insights custom metrics with .NET and .NET Core](#)
- [Application Insights API for custom events and metrics](#)
- [Application Insights telemetry data model](#)
- [Azure Monitor pricing](#)
- [Log data ingestion time in Azure Monitor](#)

- Sampling in Application Insights
- Filter and preprocess telemetry in the Application Insights SDK

Related resources

- [Architect multitenant solutions on Azure](#)
- [Architectural considerations for a multitenant solution](#)
- [Tenancy models for a multitenant solution](#)

Multitenancy and Azure Key Vault

Article • 12/11/2023

Azure Key Vault is used to manage secure data for your solution, including secrets, encryption keys, and certificates. In this article, we describe some of the features of Azure Key Vault that are useful for multitenant solutions. We then provide links to the guidance that can help you, when you're planning how you're going to use Key Vault.

Isolation models

When working with a multitenant system using Key Vault, you need to make a decision about the level of isolation that you want to use. The choice of isolation models you use depends on the following factors:

- How many tenants do you plan to have?
- Do you share your application tier between multiple tenants, do you deploy single-tenant application instances, or do you deploy separate deployment stamps for each tenant?
- Do your tenants need to manage their own encryption keys?
- Do your tenants have compliance requirements that require their secrets are stored separately from other tenants' secrets?

The following table summarizes the differences between the main tenancy models for Key Vault:

expand Expand table

Consideration	Vault per tenant, in the provider's subscription	Vault per tenant, in the tenant's subscription	Shared vault
Data isolation	High	Very high	Low
Performance isolation	Medium. High throughput might be limited, even with many vaults	High	Low
Deployment complexity	Low-medium, depending on the number of tenants	High. The tenant must correctly grant access to the provider	Low
Operational complexity	High	Low for the provider, higher for the tenant	Lowest

Consideration	Vault per tenant, in the provider's subscription	Vault per tenant, in the tenant's subscription	Shared vault
Example scenario	Individual application instances per tenant	Customer-managed encryption keys	Large multitenant solution with a shared application tier

Vault per tenant, in the provider's subscription

You might consider deploying a vault for each of your tenants within your (the service provider's) Azure subscription. This approach provides you with strong data isolation between each tenant's data, but it requires that you deploy and manage an increasing number of vaults, as you increase the number of tenants.

There's no limit to the number of vaults you can deploy into an Azure subscription. However, you should consider the following limits:

- [There are subscription-wide limits](#) on the number of requests in a time period. These limits apply regardless of the number of vaults in the subscription. So, it's important to follow our [throttling guidance](#), even when you have tenant-specific vaults.
- There's a [limit to the number of Azure role assignments that you can create within a subscription](#). When you deploy and configure large numbers of vaults in a subscription, you might approach these limits.

Vault per tenant, in the tenant's subscription

In some situations, your tenants might create vaults in their own Azure subscriptions, and they might want to grant your application access to work with secrets, certificates, or keys. This approach is appropriate when you allow *customer-managed keys* (CMKs) for encryption within your solution.

In order to access the data in your tenant's vault, the tenant must provide your application with access to their vault. This process requires that your application authenticates through their Microsoft Entra instance. One approach is to publish a [multitenant Microsoft Entra application](#). Your tenants must perform a one-time consent process. They first register the multitenant Microsoft Entra application in their own Microsoft Entra tenant. Then, they grant your multitenant Microsoft Entra application the appropriate level of access to their vault. They also need to provide you with the full resource ID of the vault that they've created. Then, your application code can use a service principal that's associated with the multitenant Microsoft Entra application in your own Microsoft Entra ID, to access each tenant's vault.

Alternatively, you might ask each tenant to create a service principal for your service to use, and to provide you with its credentials. However, this approach requires that you securely store and manage credentials for each tenant, which is a potential security liability.

If your tenants configure network access controls on their vaults, make sure you'll be able to access the vaults.

Shared vaults

You might choose to share tenants' secrets within a single vault. The vault is deployed in your (the solution provider's) Azure subscription, and you're responsible for managing it. This approach is simplest, but it provides the least data isolation and performance isolation.

You might also choose to deploy multiple shared vaults. For example, if you follow the [Deployment Stamps pattern](#), it's likely you'll deploy a shared vault within each stamp. Similarly, if you deploy a multi-region solution, you should deploy vaults into each region for the following reasons:

- To avoid cross-region traffic latency when working with the data in your vault.
- To support data residency requirements.
- To enable the use of regional vaults within other services that require same-region deployments.

When you work with a shared vault, it's important to consider the number of operations you perform against the vault. Operations include reading secrets and performing encryption or decryption operations. [Key Vault imposes limits on the number of requests](#) that can be made against a single vault, and across all of the vaults within an Azure subscription. Ensure that you follow the [throttling guidance](#). It's important to follow the recommended practices, including securely caching the secrets that you retrieve and using [envelope encryption](#) to avoid sending every encryption operation to Key Vault. When you follow these best practices, you can run high-scale solutions against a single vault.

If you need to store tenant-specific secrets, keys, or certificates, consider using a naming convention like a naming prefix. For example, you might prepend the tenant ID to the name of each secret. Then, your application code can easily load the value of a specific secret for a specific tenant.

Features of Azure Key Vault that support multitenancy

Tags

Key Vault supports tagging secrets, certificates, and keys with custom metadata, so you can use a tag to track the tenant ID for each tenant-specific secret. However, Key Vault doesn't support querying by tags, so this feature is best suited for management purposes, rather than for use within your application logic.

More information:

- [Secret tags](#)
- [Certificate tags](#)
- [Key tags](#)

Azure Policy support

If you decide to deploy a large number of vaults, it's important to ensure that they follow a consistent standard for network access configuration, logging, and access control. Consider using Azure Policy to verify the vaults have been configured according to your requirements.

More information:

- [Integrate Azure Key Vault with Azure Policy](#)
- [Azure Policy built-in definitions for Key Vault](#)

Managed HSM and Dedicated HSM

If you need to perform a large number of operations per second, and the Key Vault operation limits are insufficient, consider using either [Managed HSM](#) or [Dedicated HSM](#). Both products provide you with a reserved amount of capacity, but they're usually more costly than Key Vault. Additionally, be aware of the limits on the number of instances of these services that you can deploy into each region.

More information:

- [How do I decide whether to use Azure Key Vault or Azure Dedicated HSM?](#)
- [Is Azure Dedicated HSM right for you?](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [John Downs ↗](#) | Principal Customer Engineer, FastTrack for Azure

Other contributors:

- [Jack Lichwa ↗](#) | Principal Product Manager, Azure Key Vault
- [Arsen Vladimirskiy ↗](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Review [deployment and configuration approaches for multitenancy](#).

Considerations for using Azure Active Directory B2C in a multitenant architecture

Article • 10/10/2023

Azure Active Directory B2C (Azure AD B2C) provides business-to-consumer identity as a service. User identity is typically one of the main considerations when you design a multitenant application. Your identity solution serves as the gatekeeper to your application, ensuring that your tenants stay within the boundaries that you define for them. This article describes considerations and approaches for using Azure AD B2C in a multitenant solution.

One of the most common reasons for using Azure AD B2C is to enable [identity federation](#) for an application. Identity federation is the process of establishing trust between two identity providers so that your users can sign in with a pre-existing account. If you use Azure AD B2C, you can implement identity federation to enable your users to sign in by using their social or enterprise accounts. If you use federation, your users don't need to create a separate [local account](#) that's specific to your application.

If you're new to this topic, we recommend that you review the following resources:

- [What is Azure Active Directory B2C?](#)
- [Multitenant identity considerations](#)
- [Multitenant identity approaches](#)
- [Tenancy models](#)

ⓘ Note

In this article, two similarly named concepts are discussed: application tenants and Azure AD B2C tenants.

The term *application tenant* is used to refer to *your* tenants, which might be your customers or groups of users.

Azure AD B2C also uses the tenant concept in reference to individual directories, and the term *multitenancy* is used to refer to interactions between multiple Azure AD B2C tenants. Although the terms are the same, the concepts are not. When an Azure AD B2C tenant is referred to in this article, the full term *Azure AD B2C tenant* is used.

Identity in multitenant solutions

In multitenant solutions, it's common to combine multiple identity services to achieve different sets of requirements. Many solutions have two distinct sets of identities:

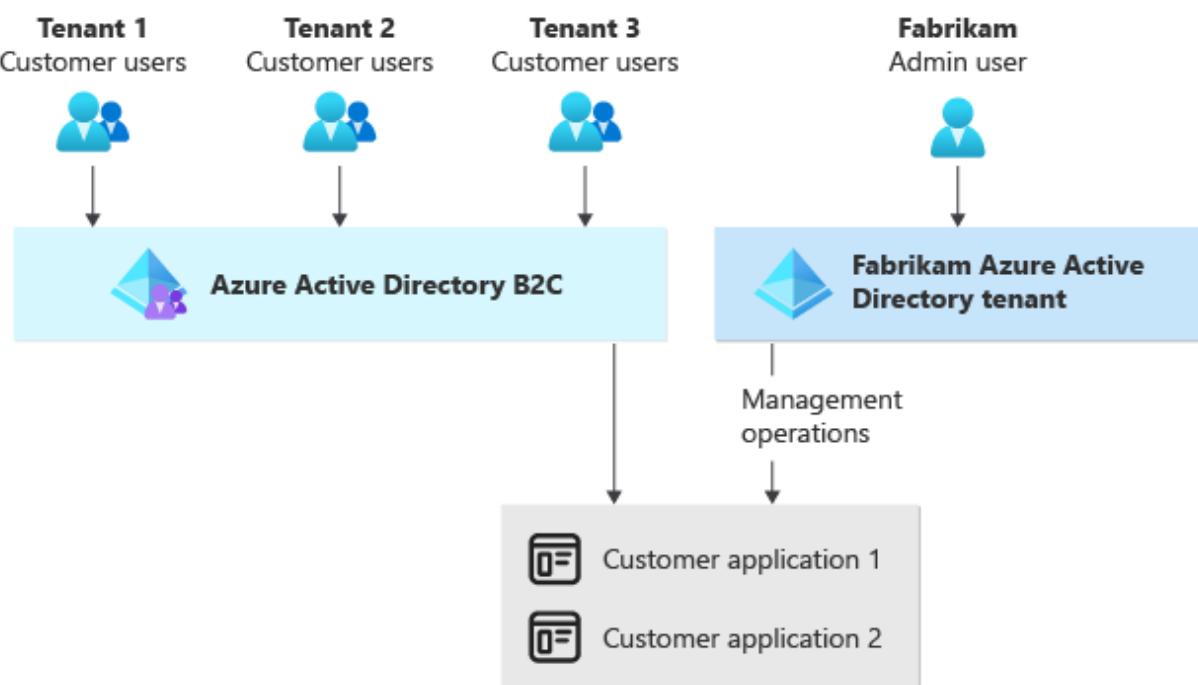
- **Customer identities**, which are for end-user accounts. They control how your tenants' users get access to your applications.
- **Internal identities**, which handle how your own team manages your solution.

These different identity types also typically use distinct identity services. Azure AD B2C is a customer identity and access management (CIAM) service that your tenants' users use to access the solution. [Microsoft Entra ID](#) (Microsoft Entra ID) is an identity and access management (IAM) service that you and your team use to manage your Azure resources and to control your application.

Consider an example multitenant solution built by Fabrikam. The solution uses a combination of the two services to meet Fabrikam's requirements:

- Fabrikam implements Azure AD B2C so that the company's customers (tenants) can sign in to applications.
- Employees of Fabrikam use their organization's Microsoft Entra directory to gain access to their solution for management and administration purposes. They use the same identities that they use for accessing other Fabrikam resources, like Microsoft Office.

The following diagram illustrates this example:



Isolation models

When you use Azure AD B2C, you need to decide how to isolate your user accounts between different application tenants.

You need to consider questions like:

- Do you need to federate sign-ins to your customer's identity providers? For example, do you need to enable federation to SAML, Microsoft Entra ID, social sign-in providers, or other sources?
- Do you or your tenants have data residency requirements?
- Does the user need to access more than one application tenant?
- Do you need complex permissions and/or role-based access control (RBAC)?
- Who signs in to your application? Different categories of users are often called *user personas*.

The following table summarizes the differences among the main tenancy models for Azure AD B2C:

Consideration	Shared Azure AD B2C tenant	Vertically partitioned Azure AD B2C tenant	One Azure AD B2C tenant per application tenant
Data isolation	Data from each application tenant is stored in a single Azure AD B2C tenant but can be accessed only by administrators	Data from each application tenant is distributed among several Azure AD B2C tenants but can be accessed only by administrators	Data from each application tenant is stored in a dedicated Azure AD B2C tenant but can be accessed only by administrators
Deployment complexity	Low	Medium to high, depending on your partitioning strategy	Very high
Limits to consider	Requests per Azure AD B2C tenant, requests per client IP address	A combination of requests, number of Azure AD B2C tenants per subscription, and number of directories for a single user, depending on your partitioning strategy	Number of Azure AD B2C tenants per subscription, maximum number of directories for a single user
Operational complexity	Low	Medium to high, depending on your partitioning strategy	Very high

Consideration	Shared Azure AD B2C tenant	Vertically partitioned Azure AD B2C tenant	One Azure AD B2C tenant per application tenant
Number of Azure AD B2C tenants required	One	Between one and n , depending on your partitioning strategy	n , where n is the number of application tenants
Example scenario	You're building a SaaS offering for consumers that has low or no data residency requirements, like a music or video streaming service.	You're building a SaaS offering, like an accounting and record keeping application for businesses. You need to support data residency requirements or a large number of custom federated identity providers.	You're building a SaaS offering, like a government record-keeping application for businesses. Your customers mandate a high degree of data isolation from other application tenants.

Shared Azure AD B2C tenant

It's generally easiest to manage a single shared Azure AD B2C tenant if your requirements allow for it. You need to maintain only one tenant long term, and this option creates the lowest overhead.

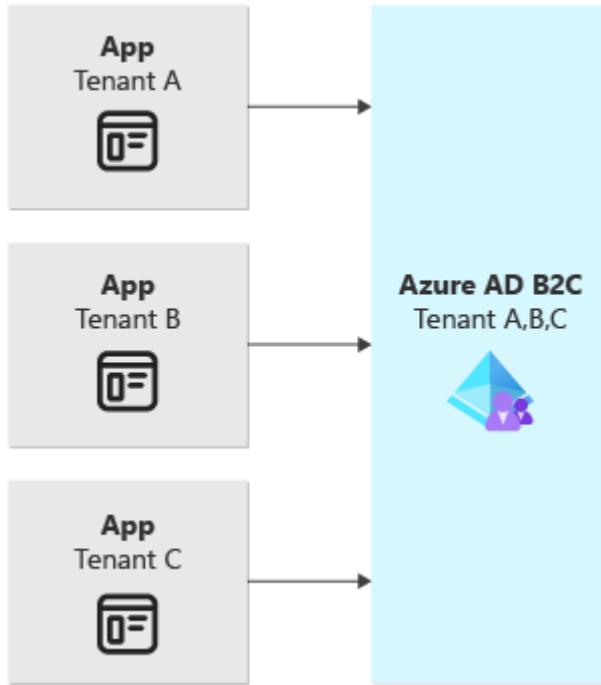
ⓘ Note

We recommend the use of a shared Azure AD B2C tenant for most scenarios.

You should consider a shared Azure AD B2C tenant when:

- You don't have data residency requirements or strict data isolation requirements.
- Your application requirements are within the Azure AD B2C [service limits](#).
- If you have federated identity providers, you can use [Home Realm Discovery](#) to automatically select a provider for a user to sign in with, or it's acceptable for users to manually select one from a list.
- You have a unified sign-in experience for all application tenants.
- Your end users need to access more than one application tenant by using a single account.

This diagram illustrates the shared Azure AD B2C tenant model:



Vertically partitioned Azure AD B2C tenants

The provisioning of vertically partitioned Azure AD B2C tenants is a strategy that's designed to minimize, when possible, the number of Azure AD B2C tenants needed. It's a middle ground between the other tenancy models. Vertical partitioning offers greater flexibility in customization for specific tenants when that's required. It doesn't, however, create the operational overhead that's associated with provisioning an Azure AD B2C tenant for every application tenant.

The deployment and maintenance requirements for this tenancy model are higher than those of a single Azure AD B2C tenant, but they're lower than they will be if you use one Azure AD B2C tenant per application tenant. You still need to design and implement a deployment and [maintenance](#) strategy for multiple tenants across your environment.

Vertical partitioning is similar to the [Data Sharding pattern](#). To vertically partition your Azure AD B2C tenants, you need to organize your application tenants into logical groups. This categorization of tenants is often called a *partitioning strategy*. Your partitioning strategy should be based on a common, stable factor of the application tenant, like region, size, or an application tenant's custom requirements. For example, if your goal is to solve your data residency requirements, you might decide to deploy an Azure AD B2C tenant for each region that hosts application tenants. Or, if you group by size, you might decide to locate most of your application tenants' identities on a single Azure AD B2C tenant, but locate your largest application tenants on their own dedicated Azure AD B2C tenants.

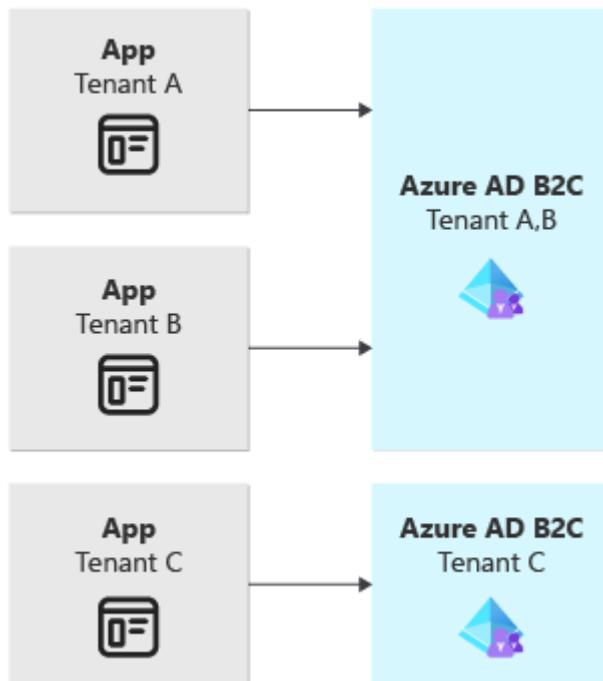


Avoid basing your partitioning strategy on factors that can change over time, because it's difficult to move users between Azure AD B2C tenants. For example, if you create a SaaS offering that has multiple SKUs or product tiers, you shouldn't partition your users based on the SKU they select, because the SKU might change if the customer upgrades their product.

You should consider provisioning your Azure AD B2C tenants by using a vertically partitioned strategy if:

- You have data residency requirements, or you need to separate your users by geography.
- You have a large number of federated identity providers and can't use [Home Realm Discovery](#) to automatically select one for a user to sign in with.
- Your application is, or can be, aware of multitenancy and knows which Azure AD B2C tenant your users need to sign in to.
- You think your larger application tenants might reach the [Azure AD B2C limits](#).
- You have a long-term strategy for deploying and [maintaining](#) a medium to large number of Azure AD B2C tenants.
- You have a strategy for sharding your application tenants among one or more Azure subscriptions to work within the limit on the number of Azure AD B2C tenants that can be deployed in an Azure subscription.

The following diagram illustrates the vertically partitioned Azure AD B2C tenant model:



One Azure AD B2C tenant per application tenant

If you provision an Azure AD B2C tenant for each application tenant, you can customize many factors for each tenant. However, this approach creates a significant increase in overhead. You need to develop a deployment and [maintenance](#) strategy for a potentially large number of Azure AD B2C tenants.

You also need to be aware of service limits. Azure subscriptions allow you to deploy only a [limited number](#) of Azure AD B2C tenants. If you need to deploy more than the limit allows, you need to consider an appropriate [subscription design](#) so you can balance your Azure AD B2C tenants across multiple subscriptions. There are other [Microsoft Entra limits](#) that apply as well, like the number of directories a single user can create and the number of directories a user can belong to.

Warning

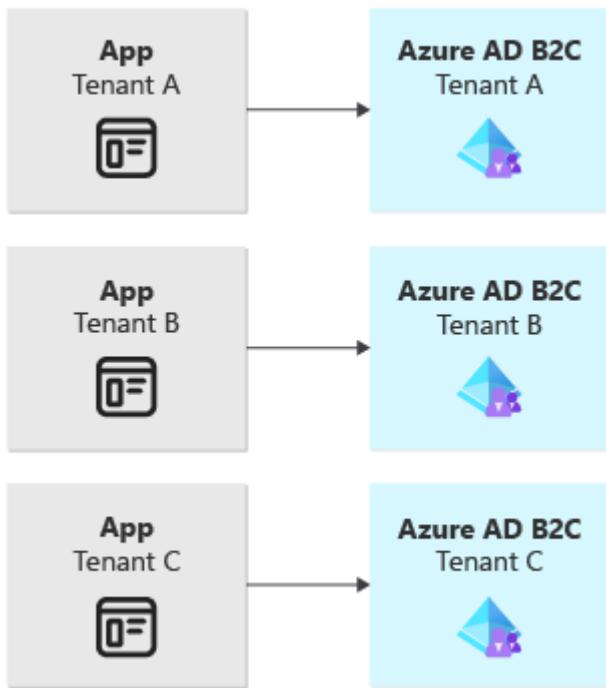
Because of the complexity of this approach, we strongly recommend that you consider the other isolation models first. This option is included here for the sake of completeness, but it's not the right approach for most use cases.

A common misconception is to assume that, if you use the [Deployment Stamps pattern](#), you need to include identity services in each stamp. That's not necessarily true, and you can often use another isolation model instead. Exercise diligence and have a clear business justification if you use this isolation model. The deployment and maintenance overhead is significant.

You should consider provisioning an Azure AD B2C tenant for every application tenant only if:

- You have strict data isolation requirements for application tenants.
- You have a long-term strategy for deploying and [maintaining](#) a large number of Azure AD B2C tenants.
- You have a strategy for sharding your customers among one or more Azure subscriptions to comply with the Azure AD B2C per-subscription tenant limit.
- Your application is, or can be, aware of multitenancy and knows which Azure AD B2C tenant your users need to sign in to.
- You need to perform custom configuration for *every* application tenant.
- Your end users don't need to access more than one application tenant via the same sign-in account.

The following diagram illustrates the use of one Azure AD B2C tenant per application tenant:



Identity federation

You need to [configure](#) each federated identity provider, either via a user flow or in a custom policy. Typically, during sign-in, users select which identity provider they want to use to authenticate. If you're using a shared tenant isolation model or have a large number of federated identity providers, consider using [Home Realm Discovery](#) to automatically select an identity provider during sign-in.

You can also use identity federation as a tool for managing multiple Azure AD B2C tenants by federating the Azure AD B2C tenants with each other. Doing so allows your application to trust a single Azure AD B2C tenant. The application doesn't need to be aware that your customers are divided among multiple Azure AD B2C tenants. This approach is most commonly used in the vertically partitioned isolation model, when your users are partitioned by region. You need to take some considerations into account if you adopt this approach. For an overview of this approach, see [Global identity solutions](#).

Home Realm Discovery

Home Realm Discovery is the process of automatically selecting a federated identity provider for a user's sign-in event. If you automatically select the user's identity provider, you don't need to prompt the user to select a provider.

Home Realm Discovery is important when you use a shared Azure AD B2C tenant and also allow your customers to bring their own federated identity provider. You might want to avoid a design in which a user needs to select from a list of identity providers.

Doing so adds complexity to the sign-in process. Also, a user might accidentally select an incorrect provider, which causes the sign-in attempt to fail.

You can configure Home Realm Discovery in various ways. The most common approach is to use the domain suffix of the user's email address to determine the identity provider. For example, say Northwind Traders is a customer of Fabrikam's multitenant solution. The email address `user@northwindtraders.com` includes the domain suffix `northwindtraders.com`, which can be mapped to the Northwind Traders federated identity provider.

For more information, see [Home Realm Discovery](#). For an example of how to implement this approach in Azure AD B2C, see the [Azure AD B2C samples GitHub repository ↗](#).

Data residency

When you provision an Azure AD B2C tenant, you select, for the purpose of [data residency](#), a region in which to deploy the tenant. This choice is important because it specifies the region in which your customer data resides when it's at rest. If you have data residency requirements for a subset of your customers, consider using the vertically partitioned strategy.

Authorization

For a strong identity solution, you need to consider *authorization* in addition to *authentication*. There are several approaches to using the Microsoft identity platform to create an authorization strategy for your application. The [AppRoles sample ↗](#) demonstrates how to use Azure AD B2C [app roles](#) to implement authorization in an application. It also describes alternative authorization approaches.

There is no single approach to authorization, and you should consider the needs of your application and your customers when you decide on an approach.

Maintenance

When you plan a multitenant deployment of Azure AD B2C, you need to think about the long-term maintenance of your Azure AD B2C resources. An Azure AD B2C tenant, like your organizational Microsoft Entra tenant, is a resource that you need to create, maintain, operate, and secure. Although the following list isn't comprehensive, you should consider the maintenance incurred in areas like these:

- **Tenant governance.** Who maintains the Azure AD B2C tenant? What elevated roles do these administrators need? How do you configure Conditional Access and MFA policies for the administrators? How do you monitor the Azure AD B2C tenant in the long term?
- **User journey configuration.** How do you deploy changes to your Azure AD B2C tenant or tenants? How do you test changes to your user flows or custom policies before you deploy them?
- **Federated identity providers.** Do you need to add or remove identity providers over time? If you allow each of your customers to bring their own identity provider, how do you manage that at scale?
- **App registrations.** Many Microsoft Entra app registrations use a [client secret](#) or [certificate](#) for authentication. How do you rotate these secrets or certificates when you need to?
- **Policy keys.** If you use custom policies, how do you rotate the policy keys when you need to?
- **User credentials.** How do you manage user information and credentials? What happens if one of your users is locked out or forgets a password and requires administrator or customer service intervention?

Remember that you need to consider these questions for every Azure AD B2C tenant that you deploy. You should also consider how your processes change when you have multiple Azure AD B2C tenants to maintain. For example, manually deploying custom policy changes to one Azure AD B2C tenant is easy, but manually deploying them to five tenants is time-consuming and risky.

Deployments and DevOps

A well-defined DevOps process can help you minimize the overhead required for maintaining your Azure AD B2C tenants. You should implement DevOps practices early in your development process. Ideally, you should try to automate all or most of your maintenance tasks, including deploying changes to your custom policies or user flows. You should also plan to create multiple Azure AD B2C tenants to progressively test changes in lower environments before you deploy them to your production tenants. Your DevOps pipelines might perform these maintenance activities. You can use the Microsoft Graph API to [programmatically manage your Azure AD B2C tenants](#).

For more information about automated deployments and management of Azure AD B2C, see the following resources.

- [Azure AD B2C operational best practices](#)
- [Deploy custom policies with Azure Pipelines](#)
- [Deploy custom policies with GitHub Actions](#)

- [Custom policy DevOps pipeline sample ↗](#)
- Graph API reference:
 - [Custom policy](#)
 - [User flow](#)
 - [App registration](#)
 - [Policy keys](#)

 **Important**

Some of the endpoints that are used to manage Azure AD B2C programmatically aren't generally available. APIs in the beta version of Microsoft Graph are subject to change at any time and are subject to prerelease terms of service.

Comparing Microsoft Entra B2B to Azure AD B2C

[Microsoft Entra B2B collaboration](#) is a feature of Microsoft Entra External ID that you can use to invite guest users into your *organizational* Microsoft Entra tenant so that you can collaborate with them. Typically, you use B2B collaboration when you need to grant an external user, like a vendor, access to resources in your Microsoft Entra tenant.

[Microsoft Entra External ID](#) is the set of approaches that you can use to interact with users outside of your organization. Azure AD B2C is one of the Microsoft Entra External ID capabilities, but it provides a different set of features than other external identities approaches. Azure AD B2C is intended to be used by the customers of your product. Your Azure AD B2C tenant is distinct from your organizational Microsoft Entra tenant.

Depending on your user personas and scenarios, you might need to use Microsoft Entra B2B, Azure AD B2C, or even both at the same time. For example, if your application needs to authenticate multiple types of users, like staff in your organization, users that work for a vendor, and customers, all within the same app, you can use Microsoft Entra B2B and Azure AD B2C together to meet this requirement.

For more information, see:

- [Use Microsoft Entra ID or Azure AD B2C?](#)
- [Comparing External Identities feature sets](#)
- [Woodgrove demo ↗](#). An example application that uses Microsoft Entra B2B and Azure AD B2C.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Landon Pierce](#) | Customer Engineer, FastTrack for Azure

Other contributors:

- [Mick Alberts](#) | Technical Writer
- [Michael Bazarewsky](#) | Senior Customer Engineer, FastTrack for Azure
- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Jelle Druyts](#) | Principal Customer Engineer, FastTrack for Azure
- [Simran Jeet Kaur](#) | Customer Engineer, FastTrack for Azure
- [LaBrina Loving](#) | Principal Customer Engineering Manager, FastTrack for Azure
- [Arsen Vladimirs](#) | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Azure AD B2C custom policy samples](#)
- [Microsoft Authentication Library \(MSAL\)](#)
- [Tutorial: Create an Azure AD B2C tenant](#)
- [Azure AD B2C authentication protocols](#)
- [Azure AD B2C limitations](#)

Related resources

- [Service-specific guidance for a multitenant solution](#)
- [Checklist for architecting and building multitenant solutions on Azure](#)
- [Architectural considerations for a multitenant solution](#)

Design patterns for multitenant SaaS applications and Azure AI Search

Article • 01/18/2024

A multitenant application is one that provides the same services and capabilities to any number of tenants who can't see or share the data of any other tenant. This article discusses tenant isolation strategies for multitenant applications built with Azure AI Search.

Azure AI Search concepts

As a search-as-a-service solution, [Azure AI Search](#) allows developers to add rich search experiences to applications without managing any infrastructure or becoming an expert in information retrieval. Data is uploaded to the service and then stored in the cloud. Using simple requests to the Azure AI Search API, the data can then be modified and searched.

Search services, indexes, fields, and documents

Before discussing design patterns, it's important to understand a few basic concepts.

When using Azure AI Search, one subscribes to a *search service*. As data is uploaded to Azure AI Search, it's stored in an *index* within the search service. There can be a number of indexes within a single service. To use the familiar concepts of databases, the search service can be likened to a database while the indexes within a service can be likened to tables within a database.

Each index within a search service has its own schema, which is defined by a number of customizable *fields*. Data is added to an Azure AI Search index in the form of individual *documents*. Each document must be uploaded to a particular index and must fit that index's schema. When searching data using Azure AI Search, the full-text search queries are issued against a particular index. To compare these concepts to those of a database, fields can be likened to columns in a table and documents can be likened to rows.

Scalability

Any Azure AI Search service in the Standard [pricing tier](#) can scale in two dimensions: storage and availability.

- *Partitions* can be added to increase the storage of a search service.
- *Replicas* can be added to a service to increase the throughput of requests that a search service can handle.

Adding and removing partitions and replicas at will allow the capacity of the search service to grow with the amount of data and traffic the application demands. In order for a search service to achieve a read [SLA](#), it requires two replicas. In order for a service to achieve a read-write [SLA](#), it requires three replicas.

Service and index limits in Azure AI Search

There are a few different [pricing tiers](#) in Azure AI Search, each of the tiers has different [limits and quotas](#). Some of these limits are at the service-level, some are at the index-level, and some are at the partition-level.

[+] [Expand table](#)

	Basic	Standard1	Standard2	Standard3	Standard3 HD
Maximum Replicas per Service	3	12	12	12	12
Maximum Partitions per Service	1	12	12	12	3
Maximum Search Units (Replicas*Partitions) per Service	3	36	36	36	36 (max 3 partitions)
Maximum Storage per Service	2 GB	300 GB	1.2 TB	2.4 TB	600 GB
Maximum Storage per Partition	2 GB	25 GB	100 GB	200 GB	200 GB
Maximum Indexes per Service	5	50	200	200	3000 (max 1000 indexes/partition)

S3 High Density

In Azure AI Search's S3 pricing tier, there's an option for the High Density (HD) mode designed specifically for multitenant scenarios. In many cases, it's necessary to support a large number of smaller tenants under a single service to achieve the benefits of simplicity and cost efficiency.

S3 HD allows for the many small indexes to be packed under the management of a single search service by trading the ability to scale out indexes using partitions for the ability to host more indexes in a single service.

An S3 service is designed to host a fixed number of indexes (maximum 200) and allow each index to scale in size horizontally as new partitions are added to the service.

Adding partitions to S3 HD services increases the maximum number of indexes that the service can host. The ideal maximum size for an individual S3HD index is around 50 - 80 GB, although there's no hard size limit on each index imposed by the system.

Considerations for multitenant applications

Multitenant applications must effectively distribute resources among the tenants while preserving some level of privacy between the various tenants. There are a few considerations when designing the architecture for such an application:

- *Tenant isolation*: Application developers need to take appropriate measures to ensure that no tenants have unauthorized or unwanted access to the data of other tenants. Beyond the perspective of data privacy, tenant isolation strategies require effective management of shared resources and protection from noisy neighbors.
- *Cloud resource cost*: As with any other application, software solutions must remain cost competitive as a component of a multitenant application.
- *Ease of Operations*: When developing a multitenant architecture, the impact on the application's operations and complexity is an important consideration. Azure AI Search has a [99.9% SLA](#).
- *Global footprint*: Multitenant applications often need to serve tenants who are distributed across the globe.
- *Scalability*: Application developers need to consider how they reconcile between maintaining a sufficiently low level of application complexity and designing the application to scale with number of tenants and the size of tenants' data and workload.

Azure AI Search offers a few boundaries that can be used to isolate tenants' data and workload.

Modeling multitenancy with Azure AI Search

In the case of a multitenant scenario, the application developer consumes one or more search services and divides their tenants among services, indexes, or both. Azure AI Search has a few common patterns when modeling a multitenant scenario:

- *One index per tenant*: Each tenant has its own index within a search service that is shared with other tenants.
- *One service per tenant*: Each tenant has its own dedicated Azure AI Search service, offering highest level of data and workload separation.
- *Mix of both*: Larger, more-active tenants are assigned dedicated services while smaller tenants are assigned individual indexes within shared services.

Model 1: One index per tenant



In an index-per-tenant model, multiple tenants occupy a single Azure AI Search service where each tenant has their own index.

Tenants achieve data isolation because all search requests and document operations are issued at an index level in Azure AI Search. In the application layer, there's the need awareness to direct the various tenants' traffic to the proper indexes while also managing resources at the service level across all tenants.

A key attribute of the index-per-tenant model is the ability for the application developer to oversubscribe the capacity of a search service among the application's tenants. If the tenants have an uneven distribution of workload, the optimal combination of tenants can be distributed across a search service's indexes to accommodate a number of highly active, resource-intensive tenants while simultaneously serving a long tail of less active tenants. The trade-off is the inability of the model to handle situations where each tenant is concurrently highly active.

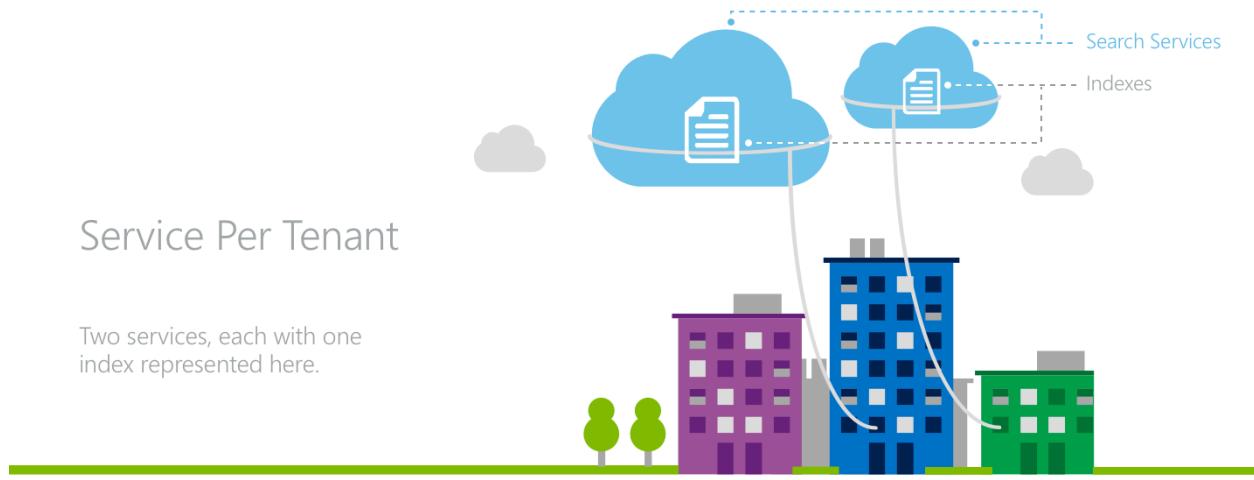
The index-per-tenant model provides the basis for a variable cost model, where an entire Azure AI Search service is bought up-front and then subsequently filled with tenants. This allows for unused capacity to be designated for trials and free accounts.

For applications with a global footprint, the index-per-tenant model might not be the most efficient. If an application's tenants are distributed across the globe, a separate service can be necessary for each region, duplicating costs across each of them.

Azure AI Search allows for the scale of both the individual indexes and the total number of indexes to grow. If an appropriate pricing tier is chosen, partitions and replicas can be added to the entire search service when an individual index within the service grows too large in terms of storage or traffic.

If the total number of indexes grows too large for a single service, another service has to be provisioned to accommodate the new tenants. If indexes have to be moved between search services as new services are added, the data from the index has to be manually copied from one index to the other as Azure AI Search doesn't allow for an index to be moved.

Model 2: One service per tenant



In a service-per-tenant architecture, each tenant has its own search service.

In this model, the application achieves the maximum level of isolation for its tenants. Each service has dedicated storage and throughput for handling search requests. Each tenant has individual ownership of API keys.

For applications where each tenant has a large footprint or the workload has little variability from tenant to tenant, the service-per-tenant model is an effective choice as resources aren't shared across various tenants' workloads.

A service per tenant model also offers the benefit of a predictable, fixed cost model. There's no up-front investment in an entire search service until there's a tenant to fill it, however the cost-per-tenant is higher than an index-per-tenant model.

The service-per-tenant model is an efficient choice for applications with a global footprint. With geographically distributed tenants, it's easy to have each tenant's service in the appropriate region.

The challenges in scaling this pattern arise when individual tenants outgrow their service. Azure AI Search doesn't currently support upgrading the pricing tier of a search service, so all data would have to be manually copied to a new service.

Model 3: Hybrid

Another pattern for modeling multitenancy is mixing both index-per-tenant and service-per-tenant strategies.

By mixing the two patterns, an application's largest tenants can occupy dedicated services while the long tail of less active, smaller tenants can occupy indexes in a shared service. This model ensures that the largest tenants have consistently high performance from the service while helping to protect the smaller tenants from any noisy neighbors.

However, implementing this strategy relies on foresight in predicting which tenants will require a dedicated service versus an index in a shared service. Application complexity increases with the need to manage both of these multitenancy models.

Achieving even finer granularity

The above design patterns to model multitenant scenarios in Azure AI Search assume a uniform scope where each tenant is a whole instance of an application. However, applications can sometimes handle many smaller scopes.

If service-per-tenant and index-per-tenant models aren't sufficiently small scopes, it's possible to model an index to achieve an even finer degree of granularity.

To have a single index behave differently for different client endpoints, a field can be added to an index, which designates a certain value for each possible client. Each time a client calls Azure AI Search to query or modify an index, the code from the client application specifies the appropriate value for that field using Azure AI Search's [filter](#) capability at query time.

This method can be used to achieve functionality of separate user accounts, separate permission levels, and even completely separate applications.

 **Note**

Using the approach described above to configure a single index to serve multiple tenants affects the relevance of search results. Search relevance scores are computed at an index-level scope, not a tenant-level scope, so all tenants' data is incorporated in the relevance scores' underlying statistics such as term frequency.

Next steps

Azure AI Search is a compelling choice for many applications. When evaluating the various design patterns for multitenant applications, consider the [various pricing tiers](#) and the respective [service limits](#) to best tailor Azure AI Search to fit application workloads and architectures of all sizes.

Multitenancy and Azure OpenAI Service

Article • 09/22/2023

Azure OpenAI provides you with access to OpenAI's powerful language models. This article describes key features of Azure OpenAI that are beneficial for multitenant solutions. Review the recommended resources to help you plan your approach and use Azure OpenAI.

Isolation models

When you have a multitenant system that uses Azure OpenAI, decide the level of isolation that you want. Determine your isolation model based on the following factors:

- How many tenants do you plan to have?
- Do you share your application tier between multiple tenants? If yes, do you deploy single-tenant application instances, or do you create separate deployments for each tenant?
- Do your tenants have compliance requirements that require access to a separate instance?

The following table summarizes the main tenancy models for Azure OpenAI.

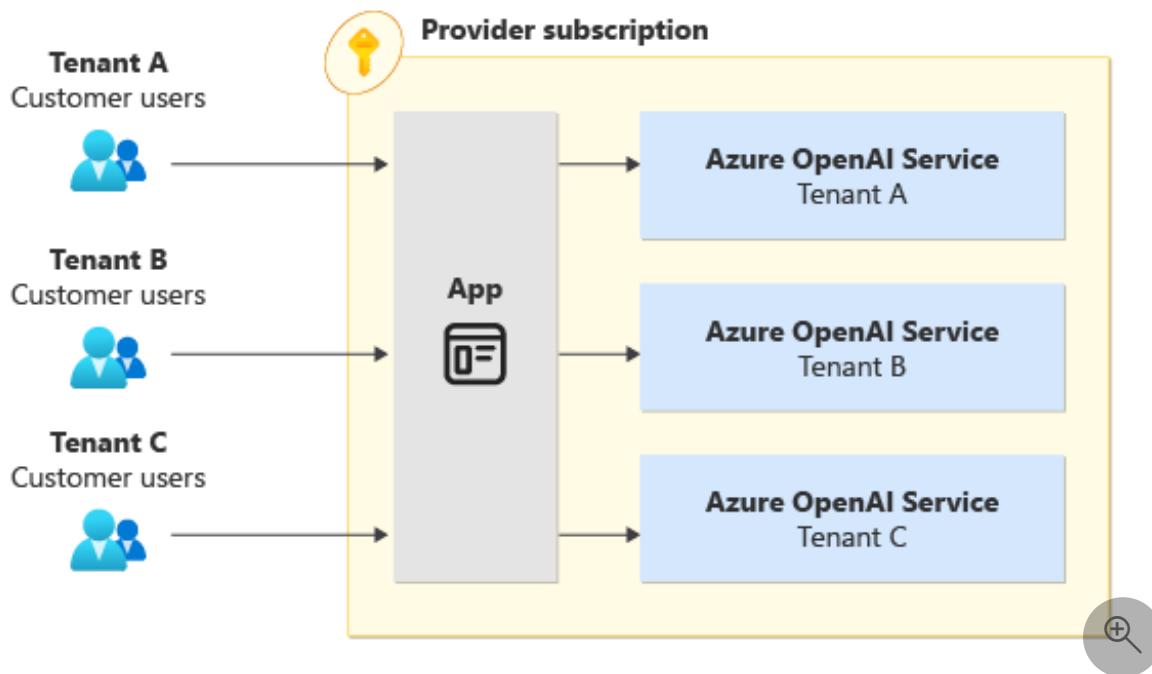
Consideration	Azure OpenAI for each tenant in the provider's subscription	Azure OpenAI for each tenant in the tenant's subscription	Shared Azure OpenAI
Data isolation	High	Very high	Low
Performance isolation	High	High	Low-medium, depending on the token per minute (TPM) usage for each tenant.
Deployment complexity	Low-medium, depending on the number of tenants.	High. The tenant must correctly grant access to the provider.	Low
Operational complexity	High	Low for the provider, higher for the tenant.	Low
Example scenario	Individual application instances for each tenant.	Tenants with specific compliance requirements or custom models.	Large multitenant solution with a shared application tier.

Azure OpenAI for each tenant in the provider's subscription

If you're a service provider, consider deploying an Azure OpenAI instance for each tenant in your Azure subscription. This approach provides data isolation for each tenant. It requires that you deploy and manage an increasing number of Azure OpenAI resources as you increase the number of tenants.

Use this approach if you have separate application deployments for each tenant, or if you need to circumvent limitations, such as the quota or request per minute. For more information, see [Azure OpenAI quotas and limits](#).

The following diagram illustrates the model for Azure OpenAI for each tenant in the provider's subscription.



Azure OpenAI for each tenant in the tenant's subscription

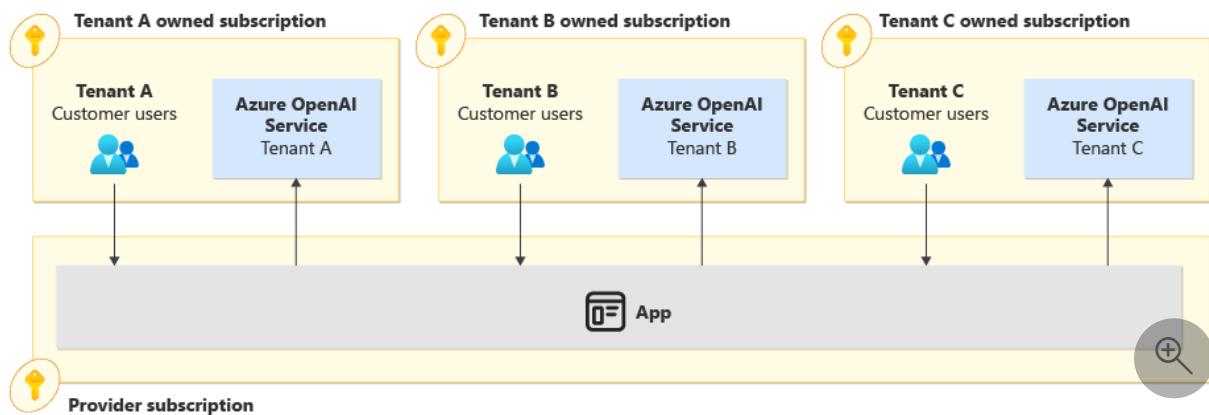
In some situations, your tenants might create the Azure OpenAI instance in their own Azure subscriptions and grant your application access to it. This approach is appropriate when tenants have specific quotas and permissions from Microsoft, such as access to the latest models, less strict filtering, or the use of provisioned throughput. You can also use this approach if the tenant has a fine-tuned model. Or if they require a component in their environment to process and send data via their customer-managed Azure OpenAI instance for processing.

To access an Azure OpenAI instance in your tenant's subscription, the tenant must provide your application with access. Your application must authenticate through their Microsoft Entra instance. One approach is to publish a [multitenant Microsoft Entra application](#). The following workflow outlines the steps of this approach:

1. The tenant registers the multitenant Microsoft Entra application in their own Microsoft Entra tenant.
2. The tenant grants the multitenant Microsoft Entra application the appropriate level of access to their Azure OpenAI resource. For example, the tenant might assign the application to the *Azure AI services user* role by using role-based access control (RBAC).
3. The tenant provides the resource ID of the Azure OpenAI resource that they create.
4. Your application code can use a service principal that's associated with the multitenant Microsoft Entra application in your own Microsoft Entra instance to access the tenant's Azure OpenAI instance.

Alternatively, you might ask each tenant to create a service principal for your service to use, and to provide you with its credentials. This approach requires that you securely store and manage credentials for each tenant, which is a potential security liability. If your tenants configure network access controls on their Azure OpenAI instance, ensure that you can access them.

The following diagram illustrates the model for Azure OpenAI for each tenant in the tenant's subscription.



Shared Azure OpenAI

You might choose to share an instance of Azure OpenAI among multiple tenants. The Azure OpenAI resource is deployed in your, or the solution provider's, Azure subscription. You're responsible for managing it. This solution is the easiest to implement, but it provides the least data isolation and performance isolation.

Sharing Azure OpenAI doesn't offer access security at the model deployment level. Other tenants can use unauthorized models. It's strongly discouraged to share an Azure OpenAI instance when you use fine-tuned models. It can expose sensitive information and allow unauthorized access to tenant-specific resources.

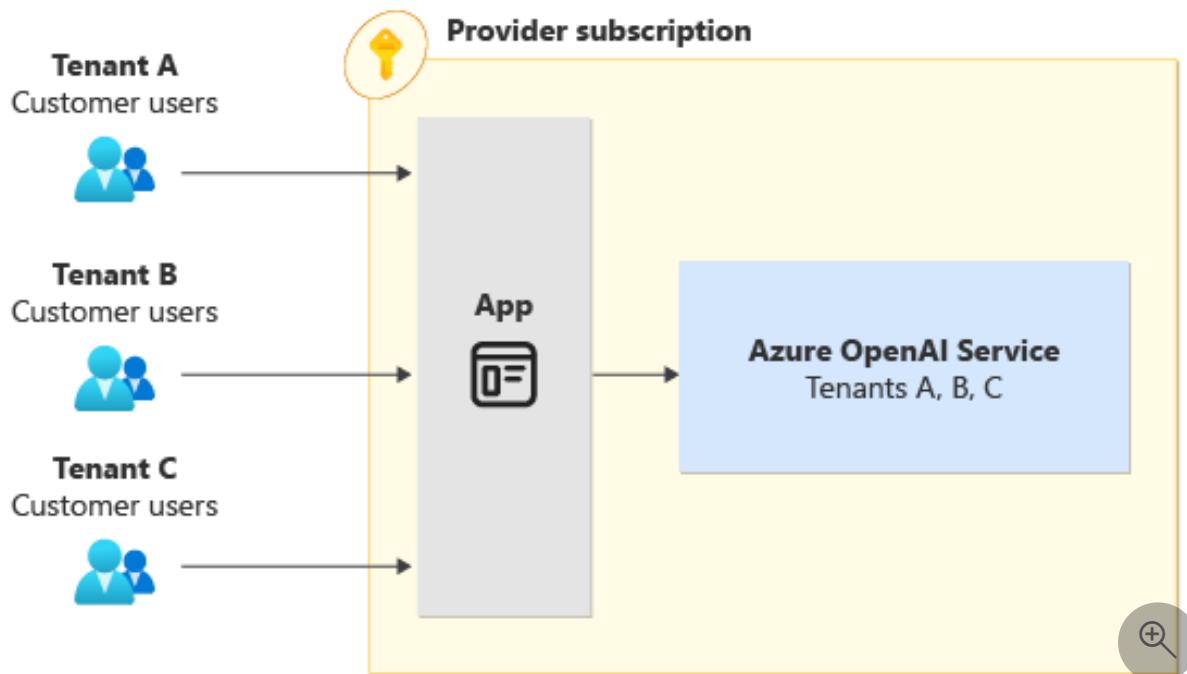
Sharing an instance of Azure OpenAI among multiple tenants can also lead to a [Noisy Neighbor](#) problem. It can cause higher latency for some tenants. You also need to make your application code multitenancy-aware. For example, if you want to charge your customers for the consumption cost of a shared Azure OpenAI instance, implement the logic to keep track of the total number of tokens for each tenant in your application.

You can also deploy multiple shared Azure OpenAI instances. For example, if you follow the [Deployment Stamps pattern](#), deploy a shared Azure OpenAI instance in each stamp. If you deploy a multiregion solution, you should deploy Azure OpenAI in each region to:

- Avoid cross-region traffic latency.
- Support data residency requirements.
- Enable the use of regional Azure OpenAI within other services that require same-region deployments.

When you have a shared Azure OpenAI instance, it's important to consider its [limits](#) and to [manage your quota](#).

The following diagram illustrates the shared Azure OpenAI model.



Shared Azure OpenAI instance with the same model for each tenant

When you use a shared Azure OpenAI instance, deploying individual instances of the same model for each tenant can offer significant benefits. This approach provides enhanced parameter customization for each deployment. It facilitates tenant-specific TPM allocation by tracking the number of tokens each model uses, which enables you to precisely cost allocate and manage each tenant's usage. This approach can optimize resource utilization to ensure that each tenant only pays for their required resources, which ensures a cost-effective solution. This approach also promotes scalability and adaptability because tenants can adjust their resource allocation based on their evolving needs and usage patterns.

Note

When you customize models for unique needs, you need to consider the approaches that are available. Every tenant might have distinct requirements and use cases. You might not use fine-tuning for most use cases. Explore other options, such as grounding. Take the time to evaluate these factors to help ensure that you choose the approach that best meets your needs.

Managed identities

Use Microsoft Entra managed identities to provide access to Azure OpenAI from other resources that are authenticated by Microsoft Entra ID. When you use managed identities, you don't need to use an Azure OpenAI API key. You can also use managed identities to grant fine-grained permissions to your Azure OpenAI identity using RBAC.

When you use managed identities, consider your isolation model. For more information, see [Azure OpenAI with managed identities](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Sofia Ferreira](#) | Software Engineer, ISV & DN CoE

Other contributors:

- [John Downs](#) | Principal Program Manager
- [Landon Pierce](#) | Customer Engineer, ISV & DN CoE

- [Paolo Salvatori](#) | Principal Customer Engineer, ISV & DN CoE
- [Daniel Scott-Raynsford](#) | Partner Tech Strategist
- [Arsen Vladimirs](#) | Principal Customer Engineer, ISV & DN CoE

To see non-public LinkedIn profiles, sign in to LinkedIn.

Related resources

- Architectural approaches for the deployment and configuration of multitenant solutions
- Architectural approaches for cost management and allocation in a multitenant solution
- Checklist for architecting and building multitenant solutions on Azure

Checklist for architecting and building multitenant solutions on Azure

Article • 04/26/2023

When you build your multitenant solution in Azure, there are many elements that you need to consider. Use this checklist as a starting point to help you design and build your multitenant solution. This checklist is a companion resource to the [Architecting multitenant solutions on Azure](#) series of articles. The checklist is structured around the business and technical considerations, and the five pillars of the [Azure Well-Architected Framework](#).

💡 Tip

After going through this checklist, take the [SaaS journey review](#) to evaluate your SaaS product by analyzing your understanding of multitenant architecture and its alignment with SaaS operation best practices.

Business considerations

- Understand what kind of solution you're creating, such as business-to-business (B2B), business-to-consumer (B2C), or your enterprise software, and [how tenants are different from users](#).
- [Define your tenants](#). Understand how many tenants you'll support initially, and your growth plans.
- [Define your pricing model](#) and ensure it aligns with your [tenants' consumption of Azure resources](#).
- Understand whether you need to separate your tenants into different [tiers](#). Tiers might have different pricing, features, performance promises, geographic locations, and so forth.
- Based on your customers' requirements, decide on the [tenancy models](#) that are appropriate for various parts of your solution.
- When you're ready, sell your B2B multitenant solution using the [Microsoft Commercial Marketplace](#).

Reliability considerations

- Review the [Azure Well-Architected Reliability checklist](#), which is applicable to all workloads.

- Understand the [Noisy Neighbor antipattern](#). Prevent individual tenants from impacting the system's availability for other tenants.
- [Design your multitenant solution](#) for the level of growth that you expect. But don't overengineer for unrealistic growth.
- Define service-level objectives (SLOs) and optionally [service-level agreements \(SLAs\)](#) for your solution. SLAs and SLOs should be based on the requirements of your tenants, as well as the [composite SLA of the Azure resources in your architecture](#).
- Test the [scale](#) of your solution. Ensure that it performs well under all levels of load, and that it scales correctly as the number of tenants increases.
- Apply [chaos engineering principles](#) to test the reliability of your solution.

Security considerations

- Apply the [Zero Trust](#) and least privilege principles in all layers of your solution.
- Ensure that you can [correctly map user requests](#) to tenants. Consider including the tenant context as part of the identity system, or by using another means, like application-level tenant authorization.
- Design for [tenant isolation](#). Continuously [test your isolation model](#).
- Ensure that your application code prevents any cross-tenant access or data leakage.
- Perform ongoing penetration testing and security code reviews.
- Understand your tenants' [compliance requirements](#), including data residency and any compliance or regulatory standards that they require you to meet.
- Correctly [manage domain names](#) and avoid vulnerabilities like [dangling DNS](#) and [subdomain takeover attacks](#).
- Follow [service-specific guidance](#) for multitenancy.

Cost Optimization considerations

- Review the [Azure Well-Architected Cost Optimization checklist](#), which is applicable to all workloads.
- Ensure you can adequately [measure per-tenant consumption](#) and correlate it with [your infrastructure costs](#).
- Avoid [antipatterns](#). Antipatterns include failing to track costs, tracking costs with unnecessary precision, real-time measurement, and using monitoring tools for billing.

Operational Excellence considerations

- Review the [Azure Well-Architected Operational Excellence checklist](#), which is applicable to all workloads.
- Use automation to manage the [tenant lifecycle](#), such as onboarding, [deployment](#), [provisioning](#), and [configuration](#).
- Understand the differences between [control plane](#) and data plane in your multitenant solution.
- Find the right balance for [deploying service updates](#). Consider both your tenants' requirements and your own operational requirements.
- Monitor the health of the overall system, as well as each tenant.
- Configure and test alerts to notify you when specific tenants are experiencing issues or are exceeding their consumption limits.
- [Organize your Azure resources](#) for isolation and scale.
- Avoid [deployment and configuration antipatterns](#). Antipatterns include running separate versions of the solution for each tenant, hardcoding tenant-specific configurations or logic, and manual deployments.

Performance Efficiency considerations

- Review the [Azure Well-Architected Performance Efficiency checklist](#), which is applicable to all workloads.
- If you use shared infrastructure, plan for how you'll mitigate [Noisy Neighbor](#) concerns. Ensure that one tenant can't reduce the performance of the system for other tenants.
- Determine how you'll scale your [compute](#), [storage](#), [networking](#), and other Azure resources to match the demands of your tenants.
- Consider each Azure resource's scale limits. [Organize your resources](#) appropriately, in order to avoid [resource organization antipatterns](#). For example, don't over-architect your solution to work within unrealistic scale requirements.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [Bohdan Cherchyk](#) | Senior Customer Engineer, FastTrack for Azure

Other contributor:

- John Downs  | Principal Customer Engineer, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Review [architectural considerations for multitenant solutions](#).
- Review [architectural approaches for multitenancy](#).
- Review [service-specific guidance for multitenancy](#).
- Review additional [resources for architects and developers of multitenant solutions](#).

Resources for architects and developers of multitenant solutions

Article • 10/10/2023

Architectures for multitenant applications

The following articles provide examples of multitenant architectures on Azure.

Architecture	Summary	Technology focus
Multitenant SaaS on Azure	Reference architecture for a multitenant SaaS scenario on Azure, which is deployed in multiple regions	Web
Use Application Gateway Ingress Controller with a multi-tenant Azure Kubernetes Service	Example for implementing multitenancy with AKS and AGIC	Kubernetes
All multitenant architectures	Lists all the architectures that include multitenancy	Multiple

Cloud design patterns

The following [cloud design patterns](#) are frequently used in multitenant architectures.

Pattern	Summary
Deployment Stamps pattern	Deploy multiple independent copies (scale units) of application components, including data stores.
Federated Identity	Delegate authentication to an external identity provider.
Gatekeeper	Protect applications and services, by using a dedicated host instance that acts as a broker between clients and the application or service, validates and sanitizes requests, and passes requests and data between them.
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.
Throttling	Control the consumption of resources that are used by an instance of an

Pattern	Summary
	application, an individual tenant, or an entire service.

Antipatterns

Consider the [Noisy Neighbor antipattern](#), in which the activity of one tenant can have a negative impact on another tenant's use of the system.

Microsoft Azure Well-Architected Framework

While the entirety of the [Azure Well-Architected Framework](#) is important for all solutions, pay special attention to the [Resiliency pillar](#). The nature of cloud hosting leads to applications that are often multitenant, use shared platform services, compete for resources and bandwidth, communicate over the internet, and run on commodity hardware. This increases the likelihood that both transient and more permanent faults will arise.

Multitenant architectural guidance

- [Architecting multitenant solutions on Azure](#) (video): This video discusses how to design, architect, and build multitenant solutions on Azure. If you're building a SaaS product or another multitenant service, there's a lot to consider when you plan for high performance, tenant isolation, and to manage deployments. This session is aimed at developers and architects who are building multitenant or SaaS applications, including startups and ISVs.
- [Azure Friday - Architecting multitenant solutions on Azure](#) (video): This video from Azure Friday discusses how to design, architect, and build multitenant software-as-a-service (SaaS) solutions on Azure.
- [Accelerate and De-Risk Your Journey to SaaS](#) (video): This video provides guidance for transitioning to the software as a service (SaaS) delivery model - whether you're starting by lifting-and-shifting an existing solution from on-premises to Azure, considering a multitenant architecture, or looking to modernize an existing SaaS web application.

Resources for Azure services

Governance and compliance

- [Organizing and managing multiple Azure subscriptions](#): It's important to consider how you manage your Azure subscriptions, as well as how you allocate tenant resources to subscriptions.
- [Cross-tenant management experiences](#): As a service provider, you can use Azure Lighthouse to manage resources, for multiple customers from within your own Microsoft Entra tenant. Many tasks and services can be performed across managed tenants, by using Azure delegated resource management.
- [Azure Managed Applications](#): In a managed application, the resources are deployed to a resource group that's managed by the publisher of the app. The resource group is present in the consumer's subscription, but an identity in the publisher's tenant has access to the resource group.

Compute

- [Best practices for cluster isolation in Azure Kubernetes Service](#): AKS provides flexibility in how you can run multitenant clusters and can isolate resources. To maximize your investment in Kubernetes, you must first understand and implement AKS multitenancy and isolation features. This best practices article focuses on isolation for cluster operators.
- [Best practices for cluster security and upgrades in Azure Kubernetes Service](#): As you manage clusters in Azure Kubernetes Service (AKS), workload and data security is a key consideration. When you run multitenant clusters using logical isolation, you especially need to secure resource and workload access.

Networking

Private Link

- [Azure Private Link Service explanation and demos from provider \(SaaS ISV\) and consumer perspectives](#) : A video that looks at the Azure Private Link service feature that enables multitenant service providers (such as independent software vendors building SaaS products). This solution enables consumers to access the provider's service using private IP addresses from the consumer's own Azure virtual networks.
- [TCP Proxy Protocol v2 with Azure Private Link Service—Deep Dive](#) : A video that presents a deep dive into TCP Proxy Protocol v2, which is an advanced feature of the Azure Private Link service. It's useful in multitenant and SaaS scenarios. The video shows you how to enable Proxy Protocol v2 in the Azure Private Link service. It also shows you how to configure an NGINX service to read the source private IP

address of the original client, rather than the NAT IP, to access the service via the private endpoint.

- [Using NGINX Plus to decode Proxy Protocol TLV linkIdentifier from the Azure Private Link service](#) : A video that looks at how to use NGINX Plus to get the TCP Proxy Protocol v2 TLV from the Azure Private Link service. The video shows how you can then extract and decode the numeric `linkIdentifier`, also called `LINKID`, of the private endpoint connection. This solution is useful for multitenant providers who need to identify the specific consumer tenant from which the connection was made.
- [SaaS Private Connectivity pattern](#) : An example solution that illustrates one approach to automate the approval of private endpoint connections, by using Azure Managed Applications.

Web

- [Claims based routing for SaaS solutions](#) : This article discusses the usage of a reverse proxy to facilitate tenant routing and mapping requests to tenants, enhancing the management of backend services in SaaS solutions.

Storage and data

- [Azure Cosmos DB and multitenant systems](#) : A blog post discussing how to build a multitenant system that uses Azure Cosmos DB.
- [Azure Cosmos DB hierarchical partition keys \(private preview\)](#) : A blog post announcing the private preview of hierarchical partition keys for Azure Cosmos DB for NoSQL. With hierarchical partition keys, also known as sub-partitioning, you can now natively partition your container with up to three levels of partition keys. This enables more optimal partitioning strategies for multitenant scenarios or workloads that would otherwise use synthetic partition keys.
- [Azure SQL Database multitenant SaaS database tenancy patterns](#): A set of articles describing various tenancy models that are available for a multitenant SaaS application, using Azure SQL Database.
- [Running 1 million databases on Azure SQL for a large SaaS provider: Microsoft Dynamics 365 and Power Platform](#) : A blog post describing how Dynamics 365 team manages databases at scale.
- [Design a multitenant database by using Azure Database for PostgreSQL Hyperscale](#)
- [Horizontal, vertical, and functional data partitioning](#): In many large-scale and multitenant solutions, data is divided into partitions that can be managed and accessed separately. Partitioning can improve scalability, reduce contention, and

optimize performance. It can also provide a mechanism for dividing data, by the usage pattern and by the tenant.

- [Data partitioning strategies by Azure service](#): This article describes some strategies for partitioning data in various Azure data stores.
- [Building multitenant applications with Azure Database for PostgreSQL Hyperscale Citus](#) (video)
- [Multitenant applications with Azure Cosmos DB](#) (video)
- [Building a multitenant SaaS with Azure Cosmos DB and Azure](#) (video): A real-world case study of how Whally, a multitenant SaaS startup, built a modern platform from scratch on Azure Cosmos DB and Azure. Whally shows the design and implementation decisions they made related to partitioning, data modeling, secure multitenancy, performance, real-time streaming from change feed to SignalR and more, all using ASP.NET Core on Azure App Services.
- [Multitenant design patterns for SaaS applications on Azure SQL Database](#) (video)

Messaging

- [Azure Event Grid domains](#): Azure Event Grid domains allow you to manage multitenant eventing architectures, at scale.
- [Service Bus sample: Cross-tenant communication using Azure Service Bus](#): Sample implementation of Azure Service Bus that shows how to communicate between a central provider and one or more customers (tenants).

Identity

- [Tenancy in Microsoft Entra ID](#): Microsoft Entra ID has its own concept of multitenancy, which refers to operating across multiple Microsoft Entra directories. When working with Microsoft Entra apps, developers can choose to configure their app to be either single-tenant or multitenant.
- [Custom-branded identity solution with Azure AD B2C](#): Azure Active Directory B2C is a customer identity access management solution that is capable of supporting millions of users and billions of authentications per day.
- [Build a multi-tenant daemon with the Microsoft identity platform endpoint](#): This sample application shows how to use the [Microsoft identity platform](#) endpoint to access the data of Microsoft business customers in a long-running, non-interactive process. It uses the OAuth2 client credentials grant to acquire an access token, which it then uses to call the Microsoft Graph and access organizational data.
- [Authenticate and authorize multitenant apps using Microsoft Entra ID](#): Learn how Microsoft Entra ID enables you to improve the functionality of cloud-native apps in multitenant scenarios.

- [Azure Architecture Walkthrough: Building a multi-tenant Azure Architecture for a B2C scenario](#): a walk through the architecture behind a multi-tenant mobile app with Azure Active Directory B2C and API Management.
- [Define and implement permissions, roles and scopes with Microsoft Entra ID in SaaS solution](#): This article covers three main concepts related to Microsoft Entra authentication & authorization, which can be used by SaaS providers. It covers Application Roles functionality, Delegated & Application permissions, and Scopes functionality.

Analytics

- [Multitenancy solutions with Power BI embedded analytics](#): When designing a multitenant application that contains Power BI Embedded, you must carefully choose the tenancy model that best fits your needs.

IoT

- [Multitenancy in IoT Hub Device Provisioning Service](#): A multitenant IoT solution will commonly assign tenant devices, by using a group of IoT hubs that are scattered across regions.

AI/ML

- [Design patterns for multitenant SaaS applications and Azure Cognitive Search](#): This document discusses tenant isolation strategies for multitenant applications that are built with Azure Cognitive Search.

Community content

Kubernetes

- [Three Tenancy Models For Kubernetes](#): Kubernetes clusters are typically used by several teams in an organization. This article explains three tenancy models for Kubernetes.
- [Understanding Kubernetes Multi Tenancy](#): Kubernetes is not a multitenant system out of the box. While it's possible to configure multitenancy, this can be challenging. This article explains Kubernetes multitenancy types.
- [Kubernetes Multi-Tenancy – A Best Practices Guide](#): Kubernetes multitenancy is a topic that more and more organizations are interested in as their Kubernetes

usage spreads out. However, since Kubernetes is not a multitenant system per se, getting multitenancy right comes with some challenges. This article describes these challenges and how to overcome them as well as some useful tools for Kubernetes multitenancy.

- [Capsule: Kubernetes multi-tenancy made simple](#) : Capsule helps to implement a multitenancy and policy-based environment in your Kubernetes cluster. It is not intended to be yet another PaaS, instead, it has been designed as a micro-services-based ecosystem with the minimalist approach, leveraging only on upstream Kubernetes.
- [Loft: Add Multi-Tenancy To Your Clusters](#) : Loft provides lightweight Kubernetes extensions for multitenancy.
- [Crossplane: The cloud native control plane framework](#) : Crossplane enables you to build control planes for your own solution, by using a Kubernetes-based approach.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [John Downs](#) | Principal Customer Engineer, FastTrack for Azure
- [Paolo Salvatori](#) | Principal Customer Engineer, FastTrack for Azure
- [Arsen Vladimirskiy](#) | Principal Customer Engineer, FastTrack for Azure
- [LaBrina Loving](#) | Principal Customer Engineering Manager, FastTrack for Azure

To see non-public LinkedIn profiles, sign in to LinkedIn.

Mission-critical baseline architecture on Azure

Azure Front Door Azure Container Registry Azure Kubernetes Service (AKS)

Azure Role-based access control

This architecture provides guidance for designing a mission critical workload on Azure. It uses cloud-native capabilities to maximize reliability and operational effectiveness. It applies the design methodology for [Well-Architected mission-critical workloads](#) to an internet-facing application, where the workload is accessed over a public endpoint and does not require private network connectivity to other company resources.

ⓘ Important



The guidance is backed by a production-grade [example implementation ↗](#) which showcases mission critical application development on Azure. This implementation can be used as a basis for further solution development in your first step towards production.

Reliability tier

Reliability is a relative concept, and for a workload to be appropriately reliable, it should reflect the business requirements surrounding it, including Service Level Objectives (SLO) and Service Level Agreements (SLA), to capture the percentage of time the application should be available.

This architecture targets an SLO of 99.99%, which corresponds to a permitted annual downtime of 52 minutes and 35 seconds. All encompassed design decisions are therefore intended to accomplish this target SLO.

💡 Tip

To define a realistic SLO, it's important to understand the SLA of all Azure components within the architecture. These individual numbers should be aggregated to determine a **composite SLA** which should align with workload targets.

Refer to [Well-Architected mission-critical workloads: Design for business requirements](#).

Key design strategies

Many factors can affect the reliability of an application, such as the ability to recover from failure, regional availability, deployment efficacy, and security. This architecture applies a set of overarching design strategies intended to address these factors and ensure the target reliability tier is achieved.

- **Redundancy in layers**

- Deploy to *multiple regions in an active-active model*. The application is distributed across two or more Azure regions that handle active user traffic.
- Utilize *Availability Zones (AZs)* for all considered services to maximize availability within a single Azure region, distributing components across physically separate data centers inside a region.
- Choose resources that support *global distribution*.

Refer to [Well-Architected mission-critical workloads: Global distribution](#).

- **Deployment stamps**

Deploy a regional stamp as a *scale unit* where a logical set of resources can be independently provisioned to keep up with the changes in demand. Each stamp also applies multiple nested scale units, such as the Frontend APIs and Background processors which can scale in and out independently.

Refer to [Well-Architected mission-critical workloads: Scale-unit architecture](#).

- **Reliable and repeatable deployments**

- Apply the *principle of Infrastructure as code (IaC)* using technologies, such as Terraform, to provide version control and a standardized operational approach for infrastructure components.
- Implement *zero downtime blue/green deployment pipelines*. Build and release pipelines must be fully automated to deploy stamps as a single operational unit, using blue/green deployments with continuous validation applied.

- Apply *environment consistency* across all considered environments, with the same deployment pipeline code across production and pre-production environments. This eliminates risks associated with deployment and process variations across environments.
- Have *continuous validation* by integrating automated testing as part of DevOps processes, including synchronized load and chaos testing, to fully validate the health of both the application code and underlying infrastructure.

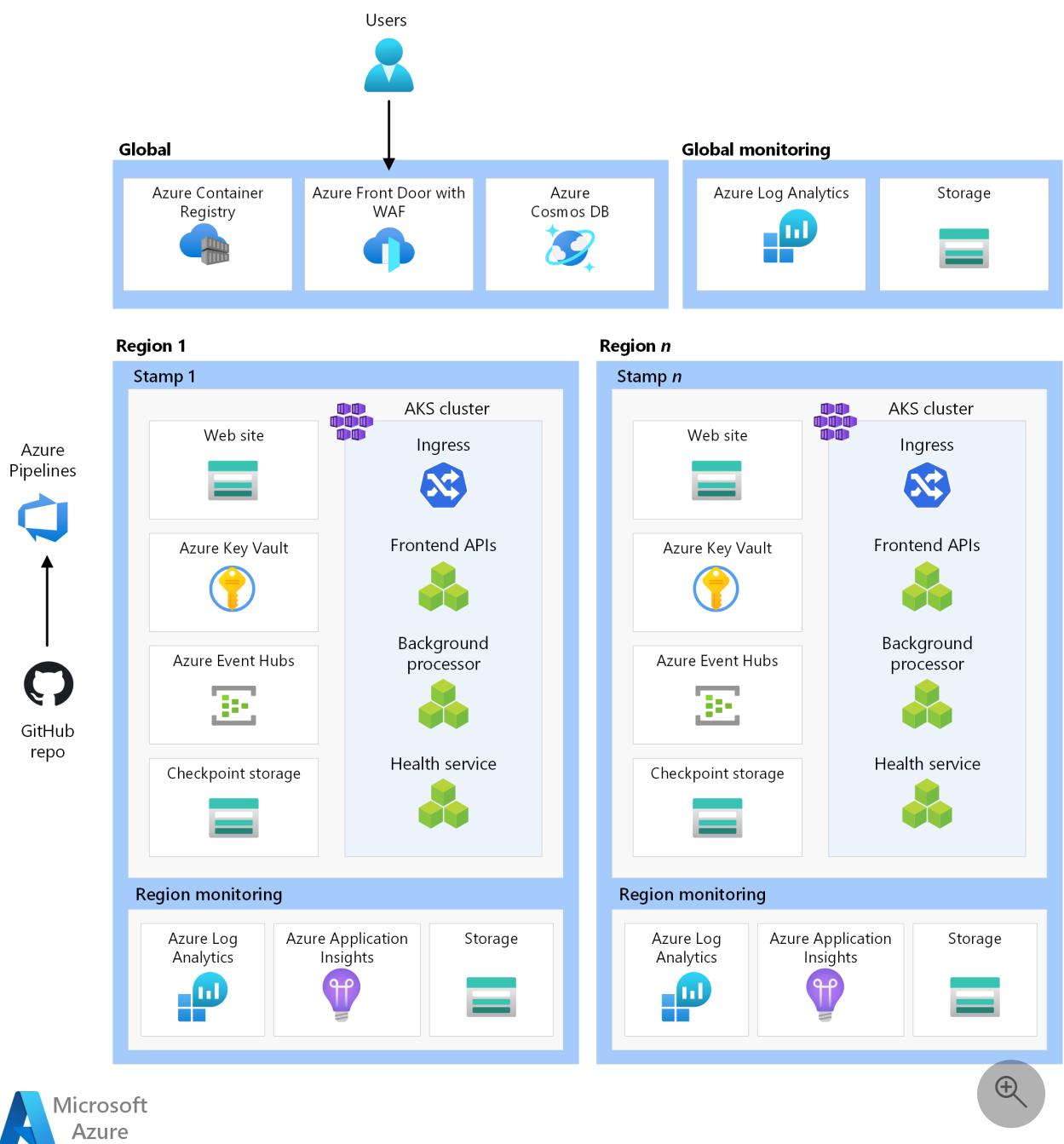
Refer to [Well-Architected mission-critical workloads: Deployment and testing](#).

- **Operational insights**

- Have *federated workspaces for observability data*. Monitoring data for global resources and regional resources are stored independently. A centralized observability store isn't recommended to avoid a single point of failure. Cross-workspace querying is used to achieve a unified data sink and single pane of glass for operations.
- Construct a *layered health model* that maps application health to a traffic light model for contextualizing. Health scores are calculated for each individual component and then aggregated at a user flow level and combined with key non-functional requirements, such as performance, as coefficients to quantify application health.

Refer to [Well-Architected mission-critical workloads: Health modeling](#).

Architecture



 Microsoft Azure



*Download a [Visio file](#) of this architecture.

The components of this architecture can be broadly categorized in this manner. For product documentation about Azure services, see [Related resources](#).

Global resources

The global resources are long living and share the lifetime of the system. They have the capability of being globally available within the context of a multi-region deployment model.

Here are the high-level considerations about the components. For detailed information about the decisions, see [Global resources](#).

Global load balancer

A global load balancer is critical for reliably routing traffic to the regional deployments with some level of guarantee based on the availability of backend services in a region. Also, this component should have the capability of inspecting ingress traffic, for example through web application firewall.

Azure Front Door is used as the global entry point for all incoming client HTTP(S) traffic, with **Web Application Firewall (WAF)** capabilities applied to secure Layer 7 ingress traffic. It uses TCP Anycast to optimize routing using the Microsoft backbone network and allows for transparent failover in the event of degraded regional health. Routing is dependent on custom health probes that check the composite health of key regional resources. Azure Front Door also provides a built-in content delivery network (CDN) to cache static assets for the website component.

Another option is Traffic Manager, which is a DNS based Layer 4 load balancer. However, failure is not transparent to all clients since DNS propagation must occur.

Refer to [Well-Architected mission-critical workloads: Global traffic routing](#).

Database

All state related to the workload is stored in an external database, **Azure Cosmos DB for NoSQL**. This option was chosen because it has the feature set needed for performance and reliability tuning, both in client and server sides. It's highly recommended that the account has multi-master write enabled.

ⓘ Note

While a multi-region-write configuration represents the gold standard for reliability, there is a significant trade-off on cost, which should be properly considered.

The account is replicated to each regional stamp and also has zonal redundancy enabled. Also, autoscaling is enabled at the container-level so that containers automatically scale the provisioned throughput as needed.

For more information, see [Data platform for mission-critical workloads](#).

Refer to [Well-Architected mission-critical workloads: Globally distributed multi-write datastore](#).

Container registry

Azure Container Registry is used to store all container images. It has geo-replication capabilities that allow the resources to function as a single registry, serving multiple regions with multi-master regional registries.

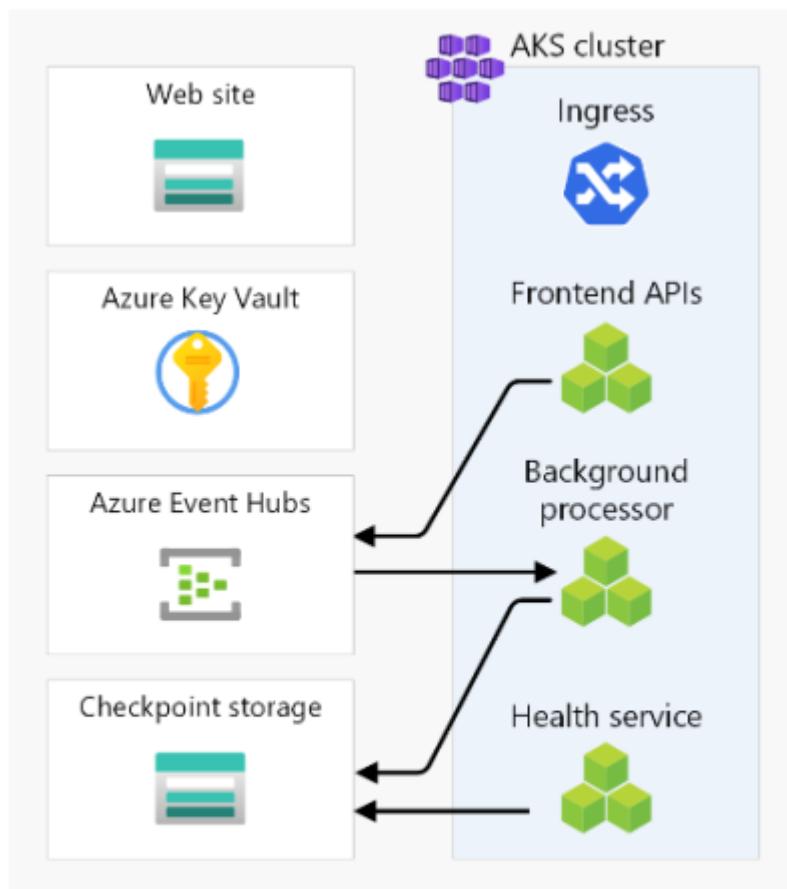
As a security measure, only allow access to required entities and authenticate that access. For example, in the implementation, admin access is disabled. So, the compute cluster can pull images only with Microsoft Entra role assignments.

Refer to [Well-Architected mission-critical workloads: Container registry](#).

Regional resources

The regional resources are provisioned as part of a *deployment stamp* to a single Azure region. These resources share nothing with resources in another region. They can be independently removed or replicated to additional regions. They, however, share [global resources](#) between each other.

In this architecture, a unified deployment pipeline deploys a stamp with these resources.



Here are the high-level considerations about the components. For detailed information about the decisions, see [Regional stamp resources](#).

Frontend

This architecture uses a single page application (SPA) that send requests to backend services. An advantage is that the compute needed for the website experience is offloaded to the client instead of your servers. The SPA is hosted as a **static website in an Azure Storage Account**.

Another choice is Azure Static Web Apps, which introduces additional considerations, such as how the certificates are exposed, connectivity to a global load balancer, and other factors.

Static content is typically cached in a store close to the client, using a content delivery network (CDN), so that the data can be served quickly without communicating directly with backend servers. It's a cost-effective way to increase reliability and reduce network latency. In this architecture, the **built-in CDN capabilities of Azure Front Door** are used to cache static website content at the edge network.

Compute cluster

The backend compute runs an application composed of three microservices and is stateless. So, containerization is an appropriate strategy to host the application. **Azure Kubernetes Service (AKS)** was chosen because it meets most business requirements and Kubernetes is widely adopted across many industries. AKS supports advanced scalability and deployment topologies. The AKS [Uptime SLA tier](#) is highly recommended for hosting mission critical applications because it provides availability guarantees for the Kubernetes control plane.

Azure offers other compute services, such as Azure Functions and Azure App Services. Those options offload additional management responsibilities to Azure at the cost of flexibility and density.

ⓘ Note

Avoid storing state on the compute cluster, keeping in mind the ephemeral nature of the stamps. As much as possible, persist state in an external database to keep scaling and recovery operations lightweight. For example in AKS, pods change frequently. Attaching state to pods will add the burden of data consistency.

Refer to [Well-Architected mission-critical workloads: Container Orchestration and Kubernetes](#).

Regional message broker

To optimize performance and maintain responsiveness during peak load, the design uses asynchronous messaging to handle intensive system flows. As a request is quickly acknowledged back to the frontend APIs, the request is also queued in a message broker. These messages are subsequently consumed by a backend service that, for instance, handles a write operation to a database.

The entire stamp is stateless except at certain points, such as this message broker. Data is queued in the broker for a short period of time. The message broker must guarantee at least once delivery. This means messages will be in the queue, if the broker becomes unavailable after the service is restored. However, it's the consumer's responsibility to determine whether those messages still need processing. The queue is drained after the message is processed and stored in a global database.

In this design, **Azure Event Hubs** is used. An additional Azure Storage account is provisioned for checkpointing. Event Hubs is the recommended choice for use cases that require high throughput, such as event streaming.

For use cases that require additional message guarantees, Azure Service Bus is recommended. It allows for two-phase commits with a client side cursor, as well as features such as a built-in dead letter queue and deduplication capabilities.

For more information, see [Messaging services for mission-critical workloads](#).

Refer to [Well-Architected mission-critical workloads: Loosely coupled event-driven architecture](#).

Regional secret store

Each stamp has its own **Azure Key Vault** that stores secrets and configuration. There are common secrets such as connection strings to the global database but there is also information unique to a single stamp, such as the Event Hubs connection string. Also, independent resources avoid a single point of failure.

Refer to [Well-Architected mission-critical workloads: Data integrity protection](#).

Deployment pipeline

Build and release pipelines for a mission-critical application must be fully automated. Therefore no action should need to be performed manually. This design demonstrates fully automated pipelines that deploy a validated stamp consistently every time. Another alternative approach is to only deploy rolling updates to an existing stamp.

Source code repository

GitHub is used for source control, providing a highly available git-based platform for collaboration on application code and infrastructure code.

Continuous Integration/Continuous Delivery (CI/CD) pipelines

Automated pipelines are required for building, testing, and deploying a mission workload in preproduction *and* production environments. **Azure Pipelines** is chosen given its rich tool set that can target Azure and other cloud platforms.

Another choice is GitHub Actions for CI/CD pipelines. The added benefit is that source code and the pipeline can be collocated. However, Azure Pipelines was chosen because of the richer CD capabilities.

Refer to [Well-Architected mission-critical workloads: DevOps processes](#).

Build Agents

Microsoft-hosted build agents are used by this implementation to reduce complexity and management overhead. Self-hosted agents can be used for scenarios that require a hardened security posture.

ⓘ Note

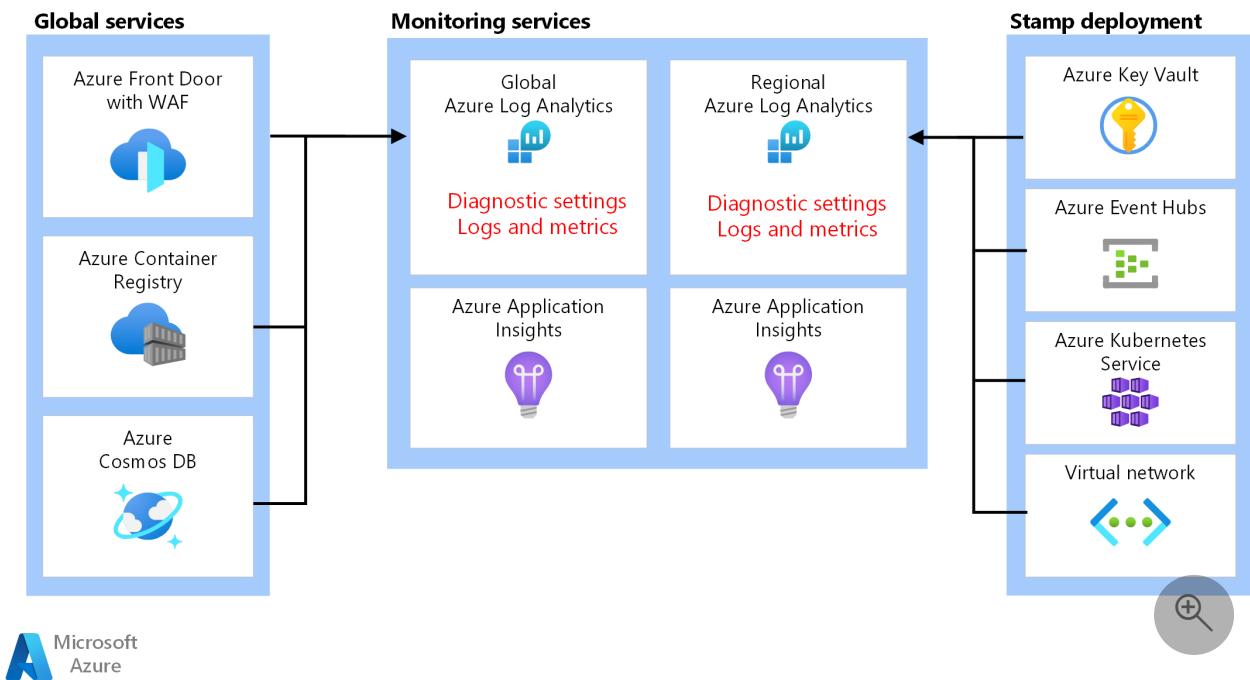
The use of self-hosted agents is demonstrated in the [Mission Critical - Connected](#) reference implementation.

Observability resources

Operational data from application and infrastructure must be available to allow for effective operations and maximize reliability. This reference provides a baseline for achieving holistic observability of an application.

Unified data sink

- **Azure Log Analytics** is used as a unified sink to store logs and metrics for all application and infrastructure components.
- **Azure Application Insights** is used as an Application Performance Management (APM) tool to collect all application monitoring data and store it directly within Log Analytics.



Monitoring data for global resources and regional resources should be stored independently. A single, centralized observability store isn't recommended to avoid a single point of failure. Cross-workspace querying is used to achieve a single pane of glass.

In this architecture, monitoring resources within a region must be independent from the stamp itself, because if you tear down a stamp, you still want to preserve observability. Each regional stamp has its own dedicated Application Insights and Log Analytics Workspace. The resources are provisioned per region but they outlive the stamps.

Similarly, data from shared services such as, Azure Front Door, Azure Cosmos DB, and Container Registry are stored in dedicated instance of Log Analytics Workspace.

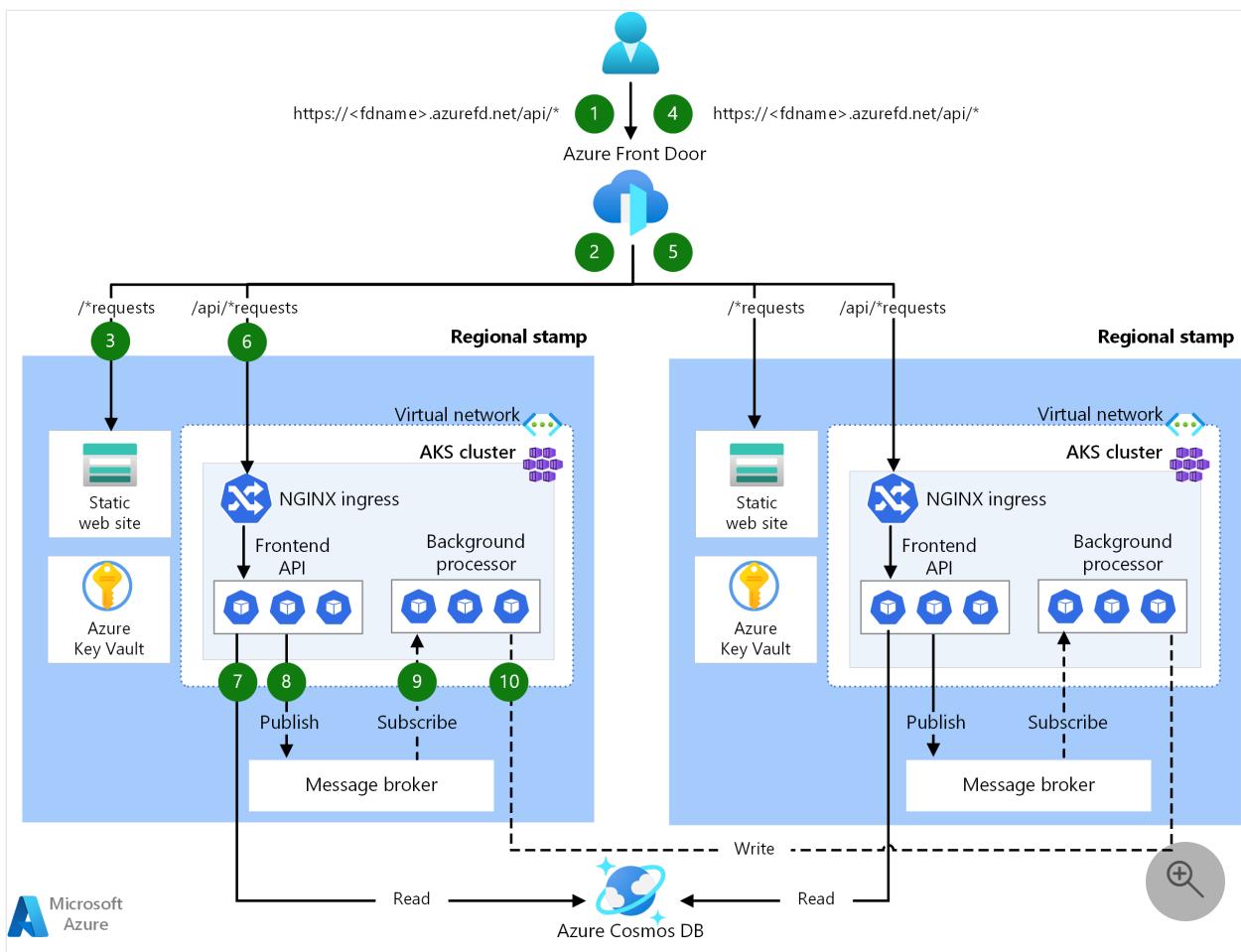
Data archiving and analytics

Operational data that isn't required for active operations are exported from Log Analytics to Azure Storage Accounts for both data retention purposes and to provide an analytical source for AIOps, which can be applied to optimize the application health model and operational procedures.

Refer to [Well-architected mission critical workloads: Predictive action and AI operations](#).

Request and processor flows

This image shows the request and background processor flow of the reference implementation.



The description of this flow is in the following sections.

Website request flow

1. A request for the web user interface is sent to a global load balancer. For this architecture, the global load balancer is Azure Front Door.
2. The WAF Rules are evaluated. WAF rules positively affect the reliability of the system by protecting against a variety of attacks such as cross-site scripting (XSS) and SQL injection. Azure Front Door will return an error to the requester if a WAF rule is violated and processing stops. If there are no WAF rules violated, Azure Front Door continues processing.
3. Azure Front Door uses routing rules to determine which backend pool to forward a request to. [How requests are matched to a routing rule](#). In this reference implementation, the routing rules allow Azure Front Door to route UI and frontend API requests to different backend resources. In this case, the pattern "/" matches the UI routing rule. This rule routes the request to a backend pool that contains storage accounts with static websites that host the Single Page Application (SPA). Azure Front Door uses the Priority and Weight assigned to the backends in the pool to select the backend to route the request. [Traffic routing methods to origin](#). Azure Front Door uses health probes to ensure that requests aren't routed to

backends that aren't healthy. The SPA is served from the selected storage account with static website.

ⓘ Note

The terms **backend pools** and **backends** in Azure Front Door Classic are called **origin groups** and **origins** in Azure Front Door Standard or Premium Tiers.

4. The SPA makes an API call to the Azure Front Door frontend host. The pattern of the API request URL is "/api/*".

Frontend API request flow

5. The WAF Rules are evaluated like in step 2.
6. Azure Front Door matches the request to the API routing rule by the "/api/*" pattern. The API routing rule routes the request to a backend pool that contains the public IP addresses for NGINX Ingress Controllers that know how to route requests to the correct service in Azure Kubernetes Service (AKS). Like before, Azure Front Door uses the Priority and Weight assigned to the backends to select the correct NGINX Ingress Controller backend.
7. For GET requests, the frontend API performs read operations on a database. For this reference implementation, the database is a global Azure Cosmos DB instance. Azure Cosmos DB has several features that make it a good choice for a mission critical workload including the ability to easily configure multi-write regions, allowing for automatic failover for reads and writes to secondary regions. The API uses the client SDK configured with retry logic to communicate with Azure Cosmos DB. The SDK determines the optimal order of available Azure Cosmos DB regions to communicate with based on the ApplicationRegion parameter.
8. For POST or PUT requests, the Frontend API performs writes to a message broker. In the reference implementation, the message broker is Azure Event Hubs. You can choose Service Bus alternatively. A handler will later read messages from the message broker and perform any required writes to Azure Cosmos DB. The API uses the client SDK to perform writes. The client can be configured for retries.

Background processor flow

9. The background processors process messages from the message broker. The background processors use the client SDK to perform reads. The client can be

configured for retries.

10. The background processors perform the appropriate write operations on the global Azure Cosmos DB instance. The background processors use the client SDK configured with retry to connect to Azure Cosmos DB. The client's preferred region list could be configured with multiple regions. In that case, if a write fails, the retry will be done on the next preferred region.

Design areas

We suggest you explore these design areas for recommendations and best practice guidance when defining your mission-critical architecture.

 Expand table

Design area	Description
Application design	Design patterns that allow for scaling, and error handling.
Application platform	Infrastructure choices and mitigations for potential failure cases.
Data platform	Choices in data store technologies, informed by evaluating required volume, velocity, variety, and veracity characteristics.
Networking and connectivity	Network considerations for routing incoming traffic to stamps.
Health modeling	Observability considerations through customer impact analysis correlated monitoring to determine overall application health.
Deployment and testing	Strategies for CI/CD pipelines and automation considerations, with incorporated testing scenarios, such as synchronized load testing and failure injection (chaos) testing.
Security	Mitigation of attack vectors through Microsoft Zero Trust model.
Operational procedures	Processes related to deployment, key management, patching and updates.

** Indicates design area considerations that are specific to this architecture.

Related resources

For product documentation on the Azure services used in this architecture, see these articles.

- [Azure Front Door](#)
- [Azure Cosmos DB](#)
- [Azure Container Registry](#)
- [Azure Log Analytics](#)
- [Azure Key Vault](#)
- [Azure Service Bus](#)
- [Azure Kubernetes Service](#)
- [Azure Application Insights](#)
- [Azure Event Hubs](#)
- [Azure Blob Storage](#)

Deploy this architecture

Deploy the reference implementation to get a complete understanding of considered resources, including how they are operationalized in a mission-critical context. It contains a deployment guide intended to illustrate a solution-oriented approach for mission-critical application development on Azure.

[Implementation: Mission-Critical Online](#)

Next steps

If you want to extend the baseline architecture with network controls on ingress and egress traffic, see this architecture.

[Architecture: Mission-critical baseline with network controls](#)

[Implementation: Mission-Critical Connected](#)

Mission-critical baseline architecture with network controls

Azure Front Door

Azure Firewall

Azure Virtual Network

Azure Kubernetes Service (AKS)

This architecture provides guidance for designing a mission critical workload that has strict network controls in place to prevent unauthorized public access from the internet to any of the workload resources. The intent is to stop attack vectors at the networking layer so that the overall reliability of the system isn't impacted. For example, a Distributed Denial of Service (DDoS) attack, if left unchecked, can cause a resource to become unavailable by overwhelming it with illegitimate traffic.

It builds on the [mission-critical baseline architecture](#), which is focused on maximizing reliability and operational effectiveness without network controls. This architecture adds features to restrict ingress and egress paths using the appropriate cloud-native capabilities, such as Azure Virtual Network(VNet) and private endpoints, Azure Private Link, Azure Private DNS Zone, and others.

It's recommended that you become familiar with the baseline before proceeding with this article.

Key design strategies

The [design strategies for mission-critical baseline](#) still apply in this use case. Here are the additional networking considerations for this architecture:

- **Control ingress traffic**

Ingress or inbound communication into the virtual network must be restricted to prevent malicious attacks.

Apply Web Application Firewall (WAF) capabilities at the global level to *stop attacks at the network edge closer to the attack source*.

Eliminate public connectivity to Azure services. One approach is to use private endpoints.

Inspect traffic before it enters the network. Network security groups (NSGs) on subnets help filter traffic by allowing or denying flow to the configured IP addresses and ports. This level of control also helps in granular logging.

- Control egress traffic

Egress traffic from a virtual network to entities outside that network must be restricted. Lack of controls might lead to data exfiltration attacks by malicious third-party services.

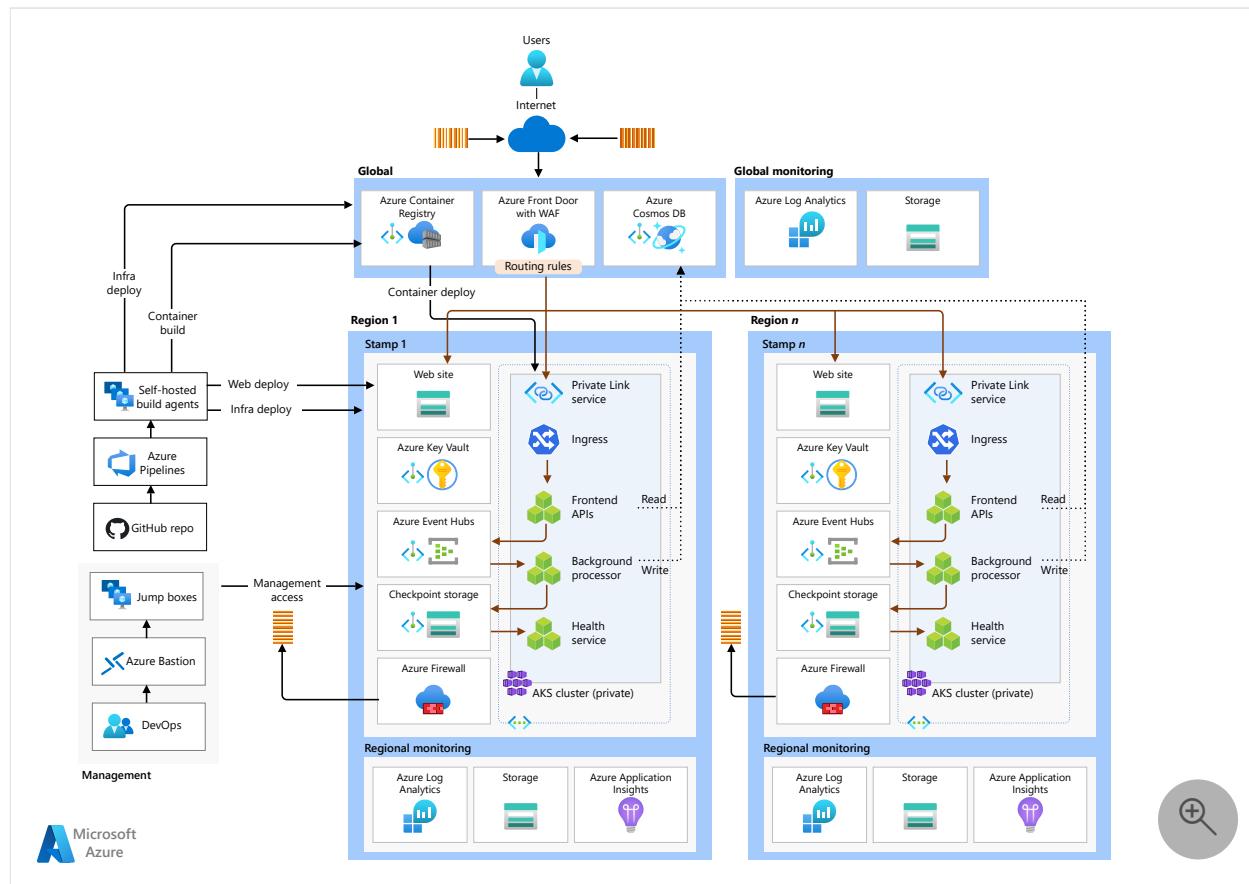
Restrict outbound traffic to the internet using Azure Firewall. Firewall can filter traffic granularly using fully qualified domain name (FQDN).

- Balance tradeoffs with security

There are significant trade-offs when security features are added to a workload architecture. You might notice some impact on performance, operational agility, and even reliability. However, *attacks, such as Denial-Of-Service (DDoS), data intrusion, and others, can target the system's overall reliability and eventually cause unavailability.*

The strategies are based on the overall guidance provided for mission-critical workloads in [Well-architected mission critical workloads](#). We suggest that you explore the [design area of networking and connectivity](#) for recommendations and best practices when defining your own mission critical architecture.

Architecture



[Download a Visio file](#) of this architecture.

The components of this architecture can be broadly categorized in this manner. For product documentation about Azure services, see [Related resources](#).

Global resources

The global resources are long living and share the lifetime of the system. They have the capability of being globally available within the context of a multi-region deployment model. For more information, see [Global resources](#).

Azure Front Door Premium SKU is used as the global load balancer for reliably routing traffic to the regional deployments, which are exposed through private endpoints.

Refer to [Well-architected mission critical workloads: Global traffic routing](#).

Azure Cosmos DB for NoSQL is still used to store state outside the compute cluster and has baseline configuration settings for reliability. Access is limited to authorized private endpoint connections.

Refer to [Well-architected mission critical workloads: Globally distributed multi-write datastore](#).

Azure Container Registry is used to store all container images with geo-replication capabilities. Access is limited to authorized private endpoint connections.

Refer to [Well-architected mission critical workloads: Container registry](#).

Regional resources

The regional resources are provisioned as part of a *deployment stamp* to a single Azure region. They are short-lived to provide more resiliency, scale, and proximity to users. These resources share nothing with resources in another region. They can be independently removed or replicated to other regions. They, however, share [global resources](#) between each other. For more information, see [Regional stamp resources](#).

Static website in an Azure Storage Account hosts a single page application (SPA) that send requests to backend services. This component has the same configuration as the [baseline frontend](#). Access is limited to authorized private endpoint connections.

Azure Virtual Networks provide secure environments for running the workload and management operations.

Internal load balancer is the application origin. Front Door uses this origin for establishing private and direct connectivity to the backend using Private Link.

Azure Kubernetes Service (AKS) is the orchestrator for backend compute that runs an application and is stateless. The AKS cluster is deployed as a private cluster. So, the Kubernetes API server isn't exposed to the public internet. Access to the API server is limited to a private network. For more information, see the [Compute cluster](#) article of this architecture.

Refer to [Well-architected mission critical workloads: Container Orchestration and Kubernetes](#).

Azure Firewall inspects and protects all egress traffic from the Azure Virtual Network resources.

Azure Event Hubs is used as the [message broker](#). Access is limited to authorized private endpoint connections.

Refer to [Well-architected mission critical workloads: Loosely coupled event-driven architecture](#).

Azure Key Vault is used as the [regional secret store](#). Access is limited to authorized private endpoint connections.

Refer to [Well-architected mission critical workloads: Data integrity protection](#).

Deployment pipeline resources

Build and release pipelines for a mission critical application must be fully automated to guarantee a consistent way of deploying a validated stamp.

GitHub is still used for source control as a highly available git-based platform.

Azure Pipelines is chosen to automate pipelines that are required for building, testing, and deploying a workload in preproduction *and* production environments.

Refer to [Well-architected mission critical workloads: DevOps processes](#).

Self-hosted Azure DevOps build agent pools are used to have more control over the builds and deployments. This level of autonomy is needed because the compute cluster and all PaaS resources are private, which requires a network level integration that is not possible on Microsoft-hosted build agents.

Observability resources

Monitoring data for global resources and regional resources are stored independently. A single, centralized observability store isn't recommended to avoid a single point of failure. For more information, see [Observability resources](#).

- **Azure Log Analytics** is used as a unified sink to store logs and metrics for all application and infrastructure components.
- **Azure Application Insights** is used as an Application Performance Management (APM) tool to collect all application monitoring data and store it directly within Log Analytics.

Refer to [Well-architected mission critical workloads: Predictive action and AI operations](#).

Management resources

A significant design change from the baseline architecture is the compute cluster. In this design the AKS cluster is private. This change requires extra resources to be provisioned to gain access.

Azure Virtual Machine Scale Sets for the private build agents and jump box instances to run tools against the cluster, such as kubectl.

Azure Bastion provides secure access to the jump box VMs and removes the need for the VMs to have public IPs.

Private endpoints for PaaS services

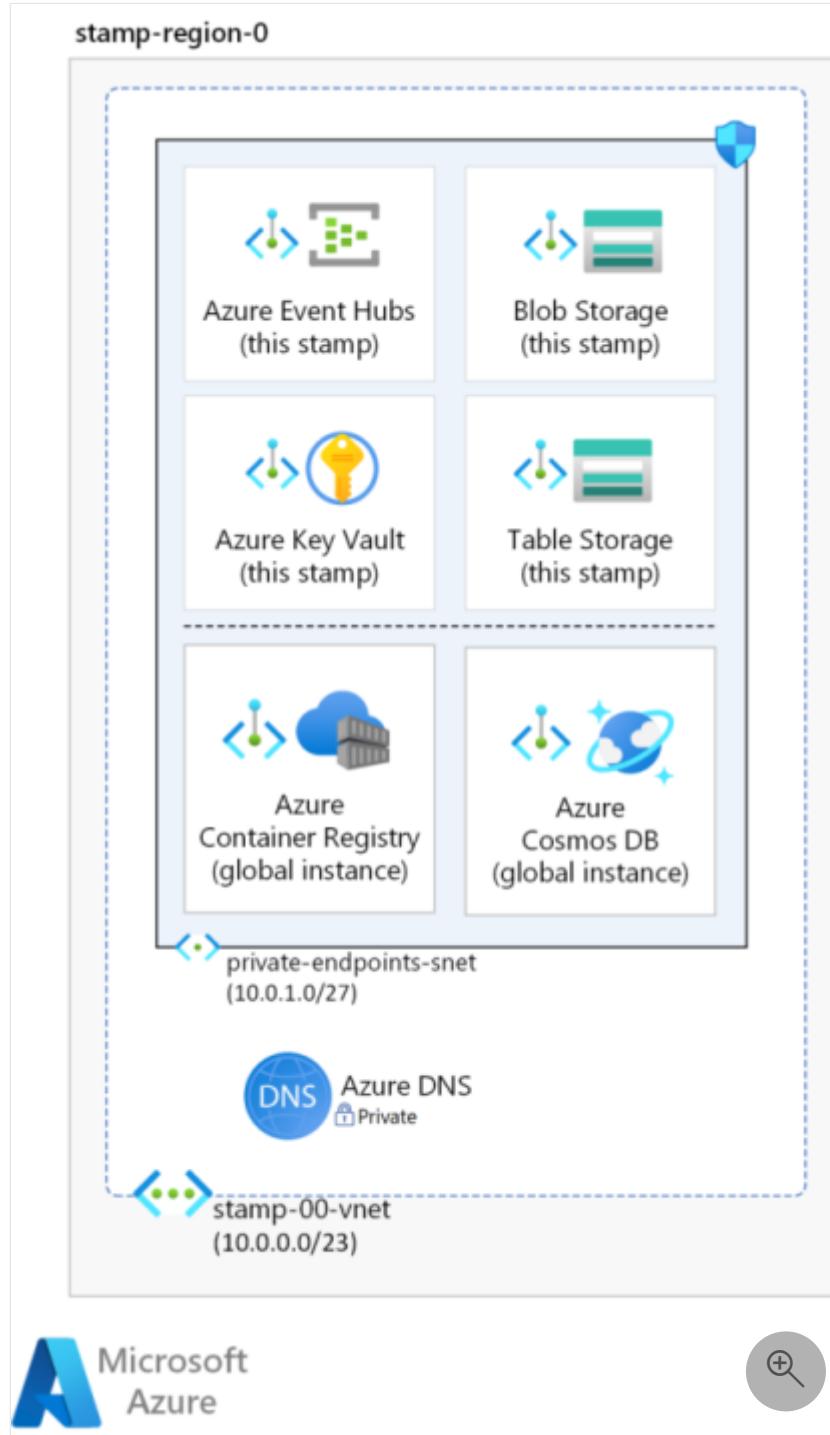
To process business or deployment operations, the application and the build agents need to reach several Azure PaaS services that are provisioned globally, within the region, and even within the stamp. In the baseline architecture, that communication is over the services' public endpoints.

In this design, those services have been protected with private endpoints to remove them from public internet access. This approach reduces the overall attack surface area to mitigate direct service tampering from unexpected sources. However, it introduces another potential point of failure and increases complexity. Carefully consider the tradeoffs with security before adopting this approach.

Private endpoints should be put in a dedicated subnet of the stamp's virtual network. Private IP addresses to the private endpoints are assigned from that subnet. Essentially,

any resource in the virtual network can communicate with the service by reaching the private IP address. Make sure the address space is large enough to accommodate all private endpoints necessary for that stamp.

To connect over a private endpoint, you need a DNS record. It's recommended that DNS records associated with the services are kept in Azure Private DNS zones serviced by Azure DNS. Make sure that the fully qualified domain name (FQDN) resolves to the private IP address.



In this architecture, private endpoints have been configured for Azure Container Registry, Azure Cosmos DB, Key Vault, Storage resources, and Event Hubs. Also, the AKS

cluster is deployed as a private cluster, which creates a private endpoint for the Kubernetes API service in the cluster's network.

There are two virtual networks provisioned in this design and both have dedicated subnets to hold private endpoints for all those services. The network layout is described in [Virtual network layout](#).

As you add more components to the architecture, consider adding more private endpoints. For example, you can add restrictions to the [observability resources](#). Both Azure Log Analytics and Azure Application Insights support the use of private endpoints. For details, see [Use Azure Private Link to connect networks to Azure Monitor](#).

They can be created on the same or different subnets within the same virtual network. There are limits to the number of private endpoints you can create in a subscription. For more information, see [Azure limits](#).

Control access to the services further by using [network security groups on the subnet](#).

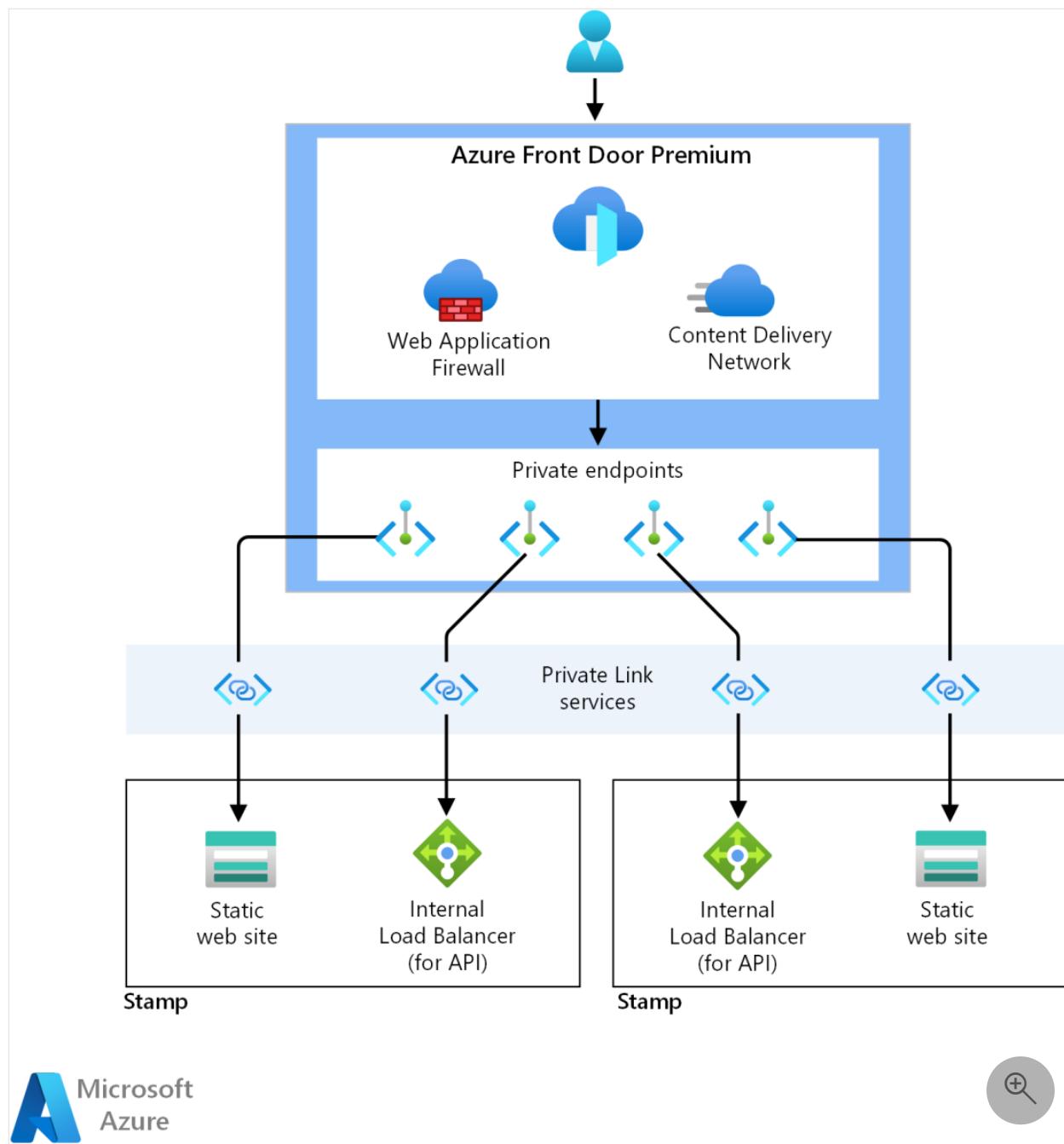
Private ingress

Azure Front Door Premium SKU is used as the global entry point for all incoming client traffic. It uses Web Application Firewall (WAF) capabilities to allow or deny traffic at the network edge. The configured WAF rules prevent attacks even before they enter the stamp virtual networks.

This architecture also takes advantage of Front Door's capability to use Azure Private Link to access application origin without the use of public IPs/endpoints on the backends. This requires an internal load balancer in the stamp virtual network. This resource is in front of the Kubernetes Ingress Controller running in the cluster. On top of this private Load Balancer, a Private Link service is created by AKS, which is used for the private connection from Front Door.

After connection is established, Private endpoints on Front Door network have direct connectivity with the load balancer and static web site in the stamp network over Private Link.

For more information, see [How Private Link works](#).



Refer to [Well-architected mission critical workloads: Application delivery services](#).

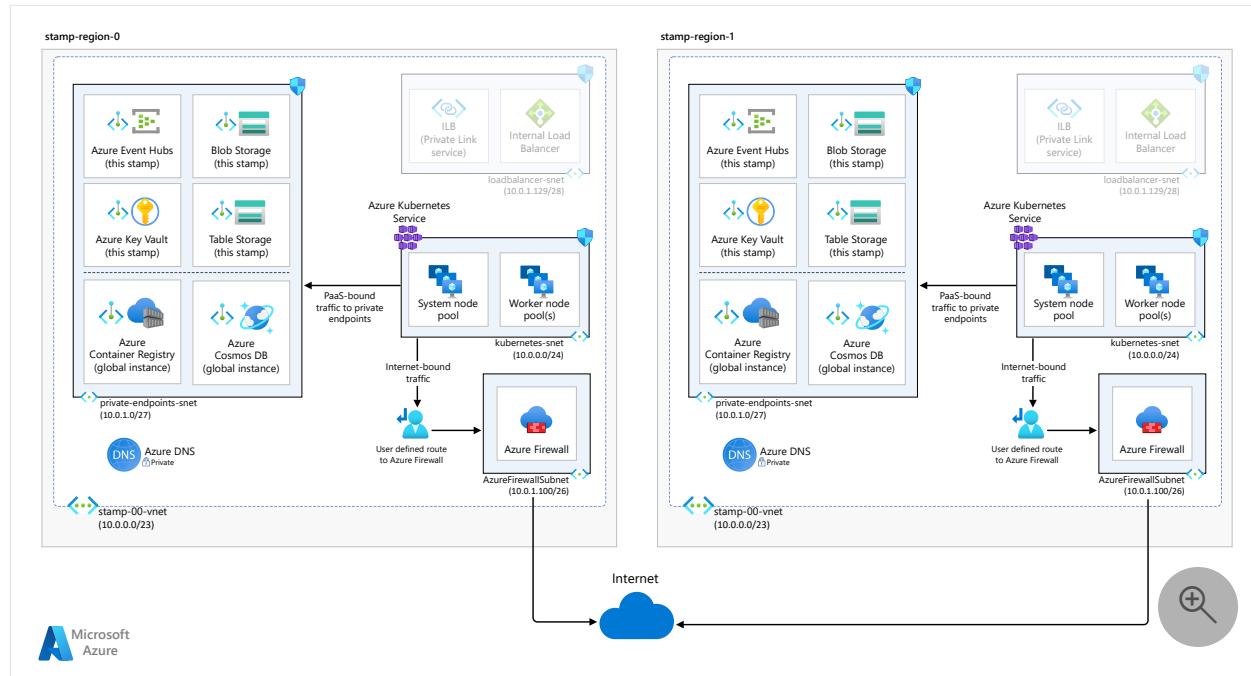
Restricted egress

Applications might require some outbound internet connectivity. Controlling that traffic provides a way to limit, monitor, and restrict egress traffic. Otherwise, unexpected inside-out access might lead to compromise and potentially an unreliable system state. Restricted egress can also solve for other security concerns, such as data exfiltration.

Using firewall and Network Security Groups (NSGs) can make sure that outbound traffic from the application is inspected and logged.

In this architecture, Azure Firewall is the single egress point and is used to inspect all outgoing traffic that originates from the virtual network. User-defined routes (UDRs) are

used on subnets that are capable of generating egress traffic, such as the application subnet.



For information about restricting outbound traffic, see [Control egress traffic for cluster nodes in Azure Kubernetes Service \(AKS\)](#).

Virtual network layout

Isolate regional resources and management resources in separate virtual networks. They have distinct characteristics, purposes, and security considerations.

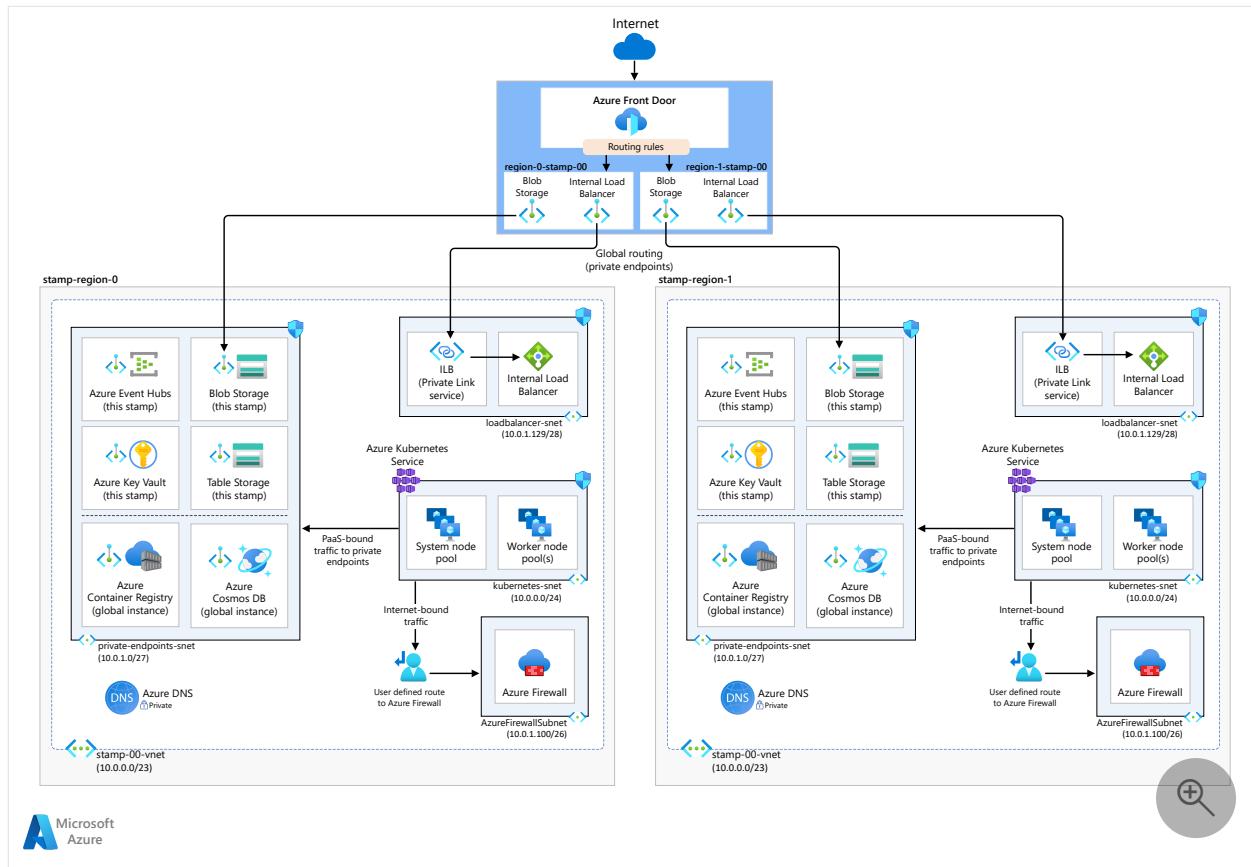
- **Type of traffic:** Regional resources, which participate in processing of business operations, need higher security controls. For example, the compute cluster must be protected from direct internet traffic. Management resources are provisioned only to access the regional resources for operations.
- **Lifetime:** The expected lifetimes of those resources are also different. Regional resources are expected to be short-lived (ephemeral). They are created as part of the deployment stamp and destroyed when the stamp is torn down. Management resources share the lifetime of the region and out live the stamp resources.

In this architecture, there are two virtual networks: stamp network and operations network. Create further isolation within each virtual network by using subnets and network security groups (NSGs) to secure communication between the subnets.

Refer to [Well-architected mission critical workloads: Isolated virtual networks](#).

Regional stamp virtual network

The deployment stamp provisions a virtual network in each region.



The virtual network is divided into these main subnets. All subnets have Network Security Groups (NSGs) assigned to block any unauthorized access from the virtual network. NSGs will restrict traffic between the application subnet and other components in the network.

- **Application subnet**

The AKS cluster node pools are isolated in a subnet. If you need to further isolate the system node pool from the worker node pool, you can place them in separate subnets.

- **Stamp ingress subnet**

The entry point to each stamp is an internal Azure Standard Load Balancer that is placed in a dedicated subnet. The Private Link service used for the private connection from Front Door is also placed here.

Both resources are provisioned as part of the stamp deployment.

- **Stamp egress subnet**

Azure Firewall is placed in a separate subnet and inspects egress traffic from application subnet by using a user-defined route (UDR).

- **Private endpoints subnet**

The application subnet will need to access the PaaS services in the regional stamp, Key Vault, and others. Also, access to global resources such as the container registry is needed. In this architecture, [all PaaS service are locked down](#) and can only be reached through private endpoints. So, another subnet is created for those endpoints. Inbound access to this subnet is secured by NSG that only allows traffic from the application.

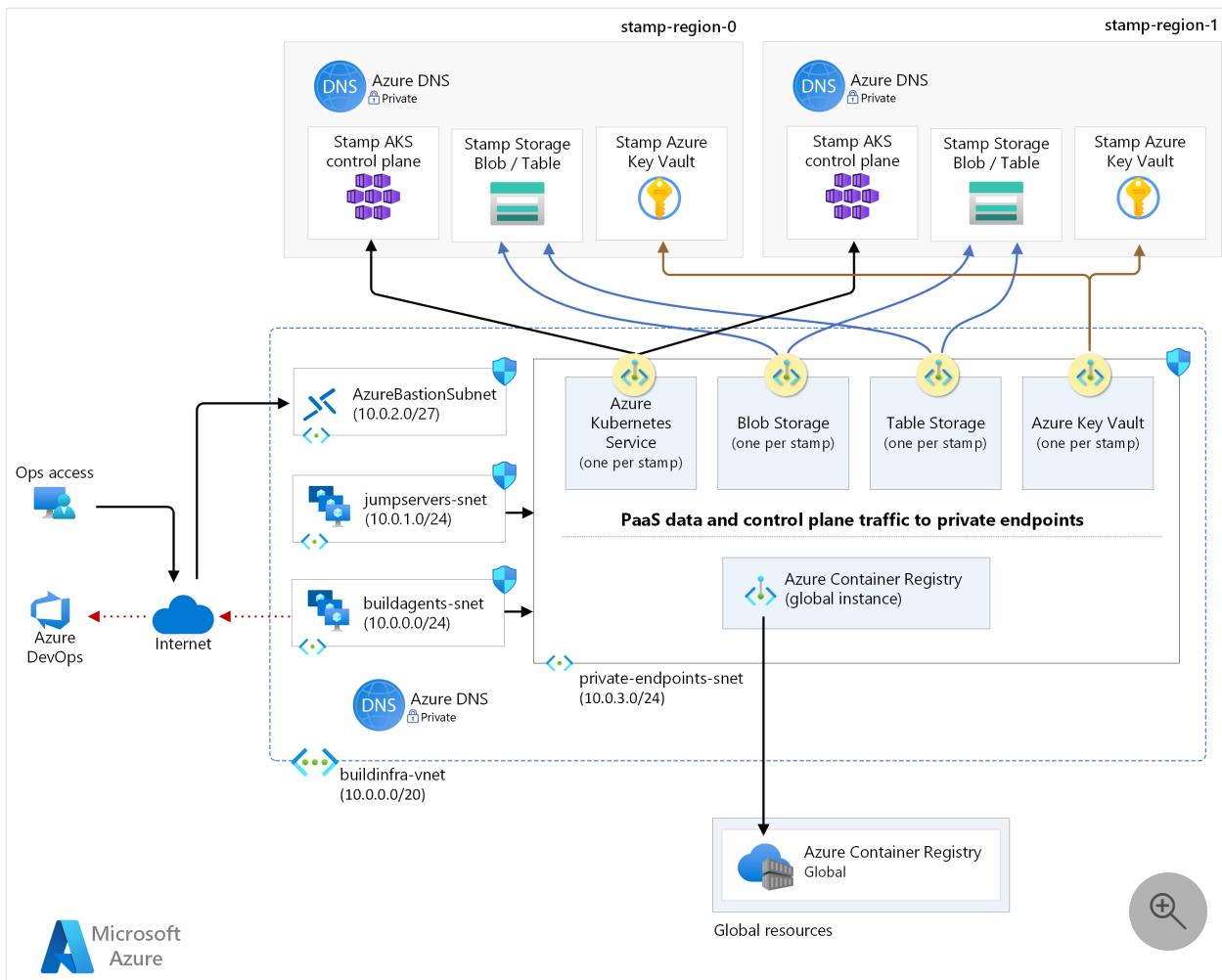
You can add further restriction by using [UDR support for private endpoints](#), so that this traffic could also egress through the stamp egress subnet.

Operations virtual network

The operational traffic is isolated in a separate virtual network. Because the AKS cluster's API service is private in this architecture, all deployment and operational traffic must also come from private resources such as self-hosted build agents and jump boxes. Those resources are deployed in a separate virtual network with direct connectivity to the application resources through their own set of private endpoints. The build agents and jump boxes are in separate subnets.

Instead of using private endpoints, an alternate approach is to use virtual network peering. However, peering adds complexity that can be hard to manage especially when virtual networks are designed to be ephemeral.

Both the build agents (and optionally jump boxes) need to access PaaS services that are located globally and within the regional stamp. Similar to the regional stamp virtual network, a dedicated subnet is created for the private endpoints to the necessary PaaS services. NSG on this subnet makes sure ingress traffic is allowed only from the management and deployment subnets.



Management operations

A typical use case is when an operator needs to access the compute cluster to run management tools and commands. The API service in a private cluster can't be accessed directly. That's why jump boxes are provisioned where the operator can run the tools. There's a separate subnet for the jump boxes.

But, those jump boxes need to be protected as well from unauthorized access. Direct access to jump boxes by opening RDP/SSH ports should be avoided. Azure Bastion is recommended for this purpose and requires a dedicated subnet in this virtual network.

⊗ Caution

Connectivity through Azure Bastion and jump boxes can have an impact on developer productivity, such as running debugging tools requires additional process. Be aware of these impacts before deciding to harden security for your mission-critical workload.

You can further restrict access to the jump box subnet by using an NSG that only allows inbound traffic from the Bastion subnet over SSH.

Deployment operations

To build deployment pipelines, you need to provision additional compute to run build agents. These resources won't directly impact the runtime availability of the workload but a reliability failure can jeopardize the ability to deploy or service your mission critical environment. So, reliability features should be extended to these resources.

This architecture uses Virtual Machine Scale Sets for both build agents and jump boxes (as opposed to single VMs). Also, network segmentation is provided through the use of subnets. Ingress is restricted to Azure DevOps.

Cost considerations

There's a significant impact on cost for mission-critical workloads. In this architecture, technology choices such as using Azure Front Door Premium SKU and provisioning Azure Firewall in each stamp will lead to increased costs. There are also added costs related to maintenance and operational resources. Such tradeoffs must be carefully considered before adopting a network-controlled version of the baseline architecture.

Deploy this architecture

The networking aspects of this architecture are implemented in the Mission-critical Connected implementation.

Implementation: Mission-critical Connected

Note

The Connected implementation is intended to illustrate a mission-critical workload that relies on organizational resources, integrates with other workloads, and uses shared services. It builds on this architecture and uses the network controls described in this article. However, the Connected scenario assumes that virtual private network or Azure Private DNS Zone already exist within the Azure landing zones connectivity subscription.

Next steps

For details on the design decisions made in this architecture, review the networking and connectivity design area for mission-critical workloads in Azure Well-architected Framework.

Related resources

For product documentation on the Azure services used in this architecture, see these articles.

- [Azure Front Door](#)
- [Azure Cosmos DB](#)
- [Azure Container Registry](#)
- [Azure Log Analytics](#)
- [Azure Key Vault](#)
- [Azure Service Bus](#)
- [Azure Kubernetes Service](#)
- [Azure Application Insights](#)
- [Azure Event Hubs](#)
- [Azure Blob Storage](#)
- [Azure Firewall](#)

Mission-critical baseline architecture in an Azure landing zone

Azure Front Door Azure Container Registry Azure Kubernetes Service (AKS)

Azure Role-based access control

This reference architecture provides guidance for deploying a mission-critical workload that uses centralized shared services, needs on-premises connectivity, and integrates with other workloads of an enterprise. This guidance is intended for a workload owner who is part of an application team in the organization.

You might find yourself in this situation when your organization wants to deploy the workload in an *Azure application landing zone* that inherits the Corp. Management group. The workload is expected to integrate with pre-provisioned shared resources in the *Azure platform landing zone* that are managed by centralized teams.

Important

What is an Azure landing zone? An application landing zone is an Azure subscription in which the workload runs. It's connected to the organization's shared resources. Through that connection, it has access to basic infrastructure needed to run the workload, such as networking, identity access management, policies, and monitoring. The platform landing zones is a collection of various subscriptions, each with a specific function. For example, the Connectivity subscription provides centralized DNS resolution, on-premises connectivity, and network virtual appliances (NVAs) that's available for use by application teams.

As a workload owner, you benefit by offloading management of shared resources to central teams and focus on workload development efforts. The organization benefits by applying consistent governance and amortizing costs across multiple workloads.

We highly recommend that you understand the concept of [Azure landing zones](#).

In this approach, maximum reliability is a shared responsibility between you and the platform team. **The centrally managed components need to be highly reliable for a mission-critical workload to operate as expected.** You must have a trusted relationship

with the platform team so that unavailability issues in the centralized services, which affect the workload, are mitigated at the platform level. But, your team is accountable for driving requirements with the platform team to achieve your targets.

This architecture builds on the [mission-critical baseline architecture with network controls](#). It's recommended that you become familiar with the [baseline architecture](#) before proceeding with this article.

ⓘ Note



The guidance is backed by a production-grade [example implementation](#) that showcases mission-critical application development on Azure. This implementation can be used as a basis for further solution development as your first step towards production.

Key design strategies

Apply these strategies on top of the [mission-critical baseline](#):

- **Critical path**

Not all components of the architecture need to be equally reliable. Critical path includes those components that must be kept functional so that the workload doesn't experience any down time or degraded performance. Keeping that path lean will minimize points of failure.

- **Lifecycle of components**

Consider the lifecycle of critical components because it has an impact on your goal of achieving near zero down time. Components can be **ephemeral** or short-lived resources that can be created and destroyed as needed; **non-ephemeral** or long-lived resources that share the lifetime with the system or region.

- **External dependencies**

Minimize external dependencies on components and processes, which can introduce points of failure. The architecture has resources owned by various teams outside your team. Those components (if critical) are in-scope for this architecture.

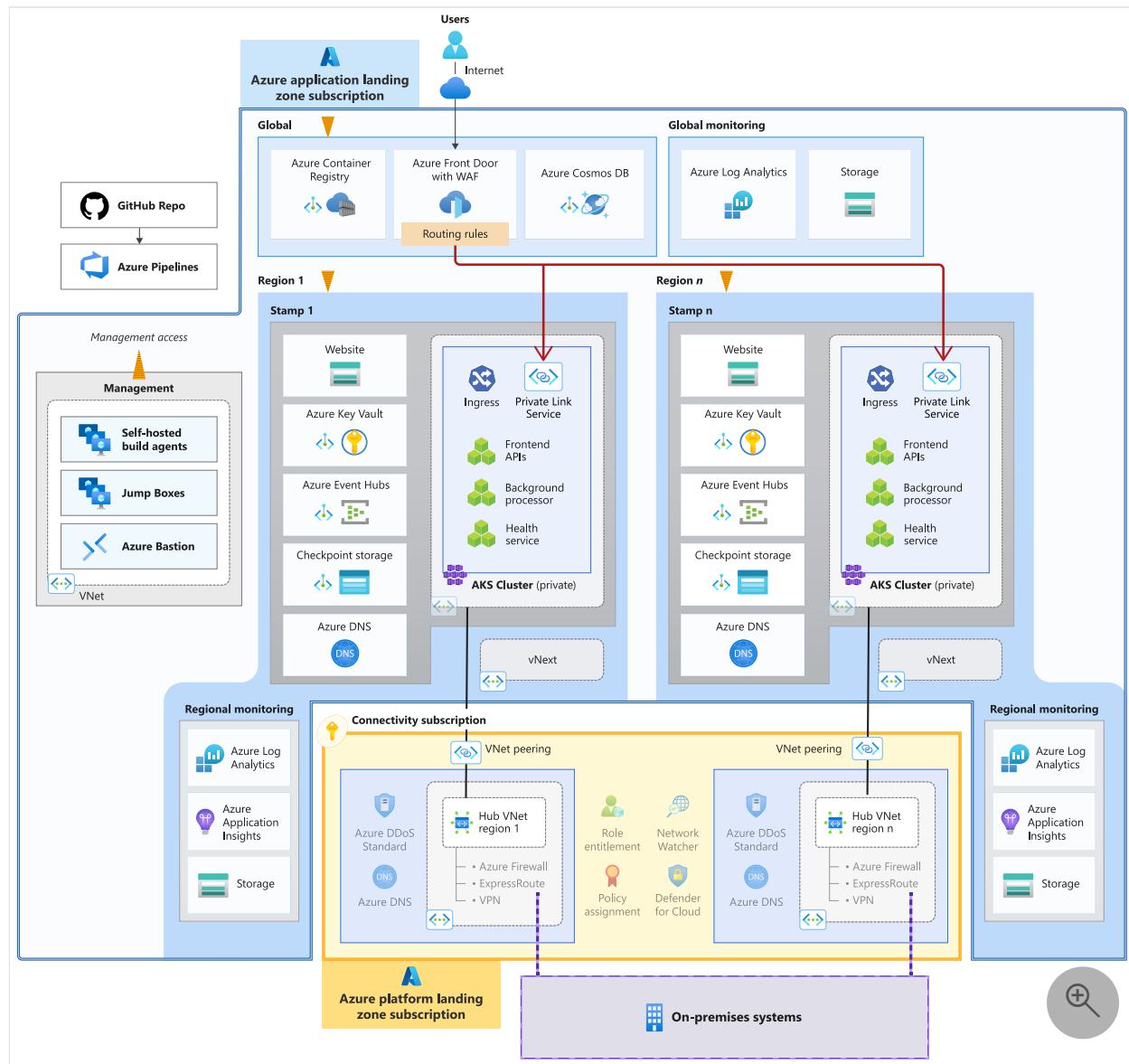
- **Subscription topology**

Azure landing zones don't imply a single subscription topology. Plan your subscription footprint in advance with your platform team to accommodate workload reliability requirements across all environments.

- Autonomous observability into the critical path

Have dedicated monitoring resources that enable the team to quickly query their data collection (especially the critical path) and act on remediations.

Architecture



Download a [Visio file](#) of this architecture.

The components of this architecture are same as the [mission-critical baseline architecture with network controls](#). The descriptions are short for brevity. If you need more information, see the linked articles. For product documentation about Azure services, see [Related resources](#).

Global resources

These resources live in the application landing zone subscription(s). Global resources are non-ephemeral and share the lifetime of the system. The Azure services and their configuration remain the same as the [baseline global resources](#).

Regional networking resources

These resources live across the platform landing zone subscriptions and the application landing zone subscription(s). The [baseline architecture](#) deploys all resources owned by you. However in this architecture, networking resources are provided through the [Connectivity subscription](#) are provisioned as part of the platform landing zone.

In this article, see the [Regional hub virtual network](#) section.

Regional stamp resources

These resources live in the application landing zone subscription(s). These resources share nothing with other resources, except the [global resources](#).

Most Azure services and their configuration remain the same as the [baseline stamp architecture](#), except for the networking resources, which are pre-provisioned by the platform team.

In this article, see the [Regional spoke virtual network](#) section.

External resources

The workload interacts with on-premises resources. Some of them are on the critical path for the workload, for example an on-premises database. These resources and communication with them is factored into the overall composite Service Level Agreement (SLA) of the workload. All communication is through the Connectivity subscription. There are other external resources that the workload might reach but they aren't considered as critical.

Deployment pipeline resources

Build and release pipelines for a mission-critical application must be fully automated to guarantee a consistent way of deploying a validated stamp. These resources remain the same as the [baseline deployment pipeline](#).

Regional monitoring resources

These resources live in the application landing zone subscription(s). Monitoring data for global resources and regional resources are stored independently. The Azure services and their configuration remain the same as the [baseline monitoring resources](#).

In this article, see the [Monitoring considerations](#) section.

Management resources

To gain access to the private compute cluster and other resources, this architecture uses private build agents and jump box virtual machines instances. Azure Bastion provides secure access to the jump boxes. The resources inside the stamps remain the same as the [baseline management resources](#).

Networking considerations

In this design, the workload is deployed in the application landing zone and needs connectivity to the federated resources in the platform landing zone. The purpose could be for accessing on-premises resources, controlling egress traffic, and so on.

Network topology

The platform team decides the network topology for the entire organization. [Hub-spoke topology](#) is assumed in this architecture. An alternative could be Azure Virtual WAN.

Regional hub virtual network

The Connectivity subscription contains a hub virtual network with resources, which are shared by the entire organization. These resources are owned and maintained by the platform team.

From a mission-critical perspective, this network is non-ephemeral, and these resources are on the critical path.

[+] [Expand table](#)

Azure resource	Purpose
Azure ExpressRoute	Provides private connectivity from on-premises to Azure infrastructure.
Azure Firewall	Acts as the NVA that inspects and restricts egress traffic.
Azure DNS	Provides cross-premises name resolution.
VPN gateway	Connects remote organization branches over the public internet to Azure infrastructure. Also acts as a backup connectivity alternative for adding resiliency.

The resources are provisioned in each region and peered to the spoke virtual network (described next). Make sure you understand and agree to the updates to NVA, firewall rules, routing rules, ExpressRoute fail over to VPN Gateway, DNS infrastructure, and so on.

ⓘ Note

A key benefit in using the federated hub is that the workload can integrate with other workloads either in Azure or cross-premises by traversing the organization-managed network hubs. This change also lowers your operational costs because a part of the responsibility is shifted to the platform team.

Regional spoke virtual network

The application landing zone has at least two pre-provisioned **Azure Virtual Networks**, per region, which are referenced by the regional stamps. These networks are non-ephemeral. One serves production traffic and the other targets the vNext deployment. This approach gives you the ability to perform [reliable and safe deployments practices](#).

Operations virtual network

Operational traffic is isolated in a separate virtual network. This virtual network is non-ephemeral and you own this network. This architecture keeps the same design as the [baseline architecture with network controls](#).

There's no peering between the operations network and spoke network. All communication is through Private Links.

Shared responsibilities

Have a clear understanding of which team is accountable for each design element of the architecture. Here are some key areas.

IP address planning

After you've estimated the size needed to run your workload, work with the platform team so that they can provision the network appropriately.

Platform team

- Provide distinct addresses for virtual networks that participate in peerings. Overlapping addresses, for example of on-premises and workload networks, can cause disruptions leading to outage.
- Allocate IP address spaces that are large enough to contain the runtime and deployments resources, handle failovers, and support scalability.

The pre-provisioned virtual network and peerings must be able to support the expected growth of the workload. You must evaluate that growth with the platform team regularly. For more information, see [Connectivity to Azure](#).

Regional spoke network subnetting

You're responsible for allocating subnets in the regional virtual network. The subnets and the resources in them are ephemeral. The address space should be large enough to accommodate potential growth.

- **Private endpoints subnet** After traffic reaches the virtual network, communication with PaaS services within the network, is restricted by using private endpoints, just like the [baseline architecture with network controls](#). These endpoints are placed in a dedicated subnet. Private IP addresses to the private endpoints are assigned from that subnet.
- **Cluster subnet** The scalability requirements of the workload influence how much address space should be allocated for the subnets. As AKS nodes and pods scale out, IP addresses are assigned from this subnet.

Network segmentation

Maintain proper segmentation so that your workload's reliability isn't compromised by unauthorized access.

This architecture uses Network Security Groups (NSGs) to restrict traffic across subnets and the Connectivity subscription. NSGs use ServiceTags for the supported services.

Platform team

- Enforce the use of NSGs through Azure Network Manager Policies.
- Be aware of the workload design. There isn't any direct traffic between the stamps. Also there aren't inter-region flows. If those paths are needed, traffic must flow through the Connectivity subscription.
- Prevent unnecessary hub traffic originating from other workloads into the mission-critical workload.

Egress traffic from regional stamps

All outgoing traffic from each regional spoke network is routed through the centralized Azure Firewall in the regional hub network. It acts as the next hop that inspects and then allows or denies traffic.

Platform team

- Create UDRs for that custom route.
- Assign Azure policies that will block the application team from creating subnets that don't have the new route table.
- Give adequate role-based access control (RBAC) permissions to the application team so that they can extend the routes based on the requirements of the workload.

Multi-region redundancy

Your mission-critical workloads must be deployed in multiple regions to withstand regional outages. Work with the platform team to make sure the infrastructure is reliable.

Platform team

- Deploy centralized networking resources per region. The mission-critical design methodology requires regional isolation.
- Work with the application team to uncover hidden regional dependencies so that a degraded platform resource in one region doesn't impact workloads in another region.

DNS resolution

The Connectivity subscription provides private DNS zones. However, that centralized approach might not factor in the DNS needs that might be specific to your use case. Provision your own DNS zones and link to the regional stamp. If the application team doesn't own DNS, then management of private links becomes challenging for global resources, such as Azure Cosmos DB.

Platform team

- Delegate the Azure Private DNS zones to the application team to cover their use cases.
- For the regional hub network, set the DNS servers value to Default (Azure-provided) to support private DNS zones managed by the application team.

Environment design considerations

It's a general practice to place workloads in separate environments for **production**, **pre-production**, and **development**. Here are some key considerations.

How is isolation maintained?

The production environment *must* be isolated from other environments. Lower environments should also maintain a level of isolation. Avoid sharing application-owned resources between environments.

One production environment is required for global, regional, and stamp resources owned by the application team. Pre-production environments, such as staging and integration, are needed to make sure the application is tested in an environment that simulates production, as much as possible. Development environment should be a scaled down version of production.

What is the expected lifecycle?

Pre-production environments can be destroyed after validation tests are completed. It's recommended that development environments share the lifetime of a feature and are destroyed when development is complete. Those actions done by continuous integration/continuous deployment (CI/CD) automation.

What are the tradeoffs?

Consider the tradeoffs between isolation of environments, complexity of management, and cost optimization.

💡 Tip

All environments should take dependencies on production instances of external resources including platform resources. For example, a production regional hub in the Connectivity subscription. You'll be able to minimize the delta between pre-production and production environments.

Subscription topology for workload infrastructure

Subscriptions are given to you by the platform team. Depending on the **number of environments**, you'll request several subscriptions for just one workload. Depending on the **type of environment**, some environments might need dedicated subscriptions while other environments might be consolidated into one subscription.

Regardless, work with the platform team to design a topology that meets the overall reliability target for the workload. There's benefit to sharing the platform-provided resources between environments in the same subscription because it will reflect the production environment.

ⓘ Note

Using multiple subscriptions to contain the environments can achieve the required level of isolation. Landing zone subscriptions are inherited from the same management group. So, consistency with production is ensured for testing and validation.

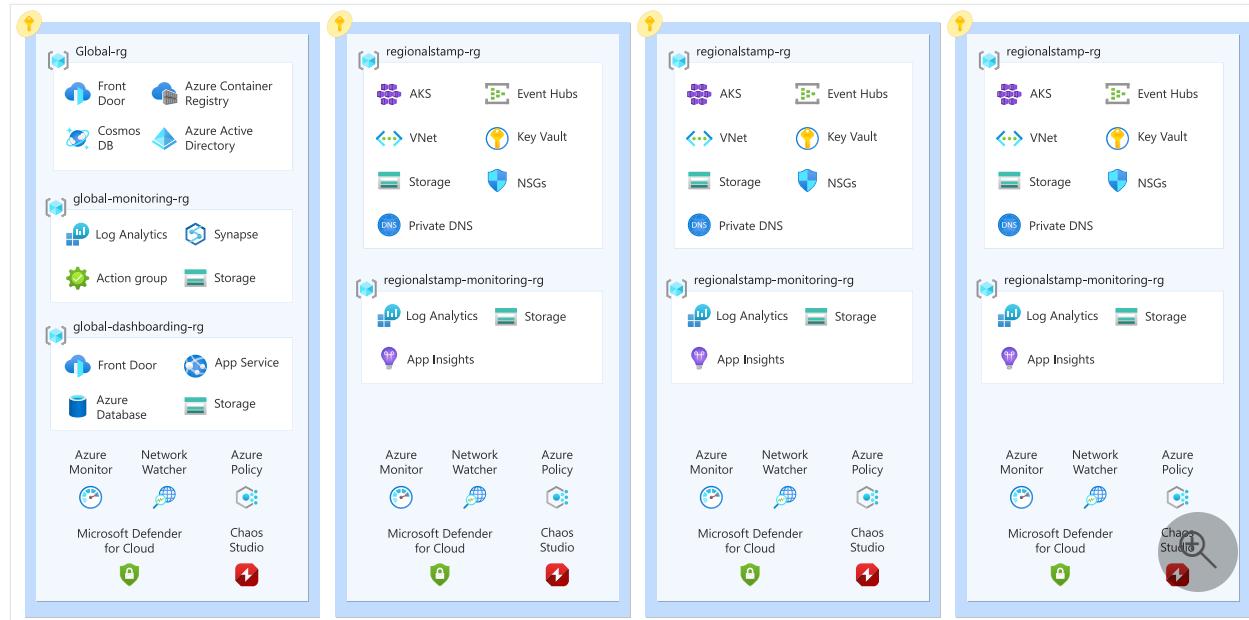
Production subscription

There might be resource limits on the subscription given to you. If you colocate all those resources in one subscription, you may reach those limits. Based on your scale units and expected scale, consider a more distributed model. For example,

- One subscription that contains both Azure DevOps build agents and global resources.

- One subscription, per region. It contains the regional resources, the stamp resources, and jump boxes for the regional stamp(s).

Here's an example subscription topology used in this architecture.



Pre-production subscription

At least one subscription is required. It can run many independent environments, however, having multiple environments in dedicated subscriptions is recommended. This subscription may also be subject to resource limits like the production subscription, described above.

Development subscription

At least one subscription is required. This subscription might be subject to resource limits if runs all independent environments. So, you may request multiple subscriptions. Having individual environments in their dedicated subscriptions is recommended.

Try to match the production topology as much as possible.

Deployment considerations

Failed deployments or erroneous releases are common causes for application outages. Test your application (and new features) thoroughly as part of the application lifecycle. When deploying the workload in a landing zone, you'll need to integrate your design with the platform-provided resources and governance.

Refer to: [Well-architected mission-critical workloads: Deployment and testing](#).

Deployment infrastructure

Reliability of the deployment infrastructure, such as build agents and their network, is often as important as the runtime resources of the workload.

This architecture uses private build agents to prevent unauthorized access that can impact the application's availability.

Maintaining isolation between deployment resources is highly recommended. A deployment infrastructure should be bound to your workload subscription(s) for that environment. If you're using multiple environments in pre-production subscriptions, then create further separation by limiting access to only those individual environments. Per-region deployment resources could be considered to make the deployment infrastructure more reliable.

Zero-downtime deployment

Updates to the application can cause outages. Enforcing consistent deployments will boost reliability. These approaches are recommended:

- Have fully automated deployment pipelines.
- Deploy updates in pre-production environments on a clean stamp.

For more information, see [Mission-critical deployment and testing guidelines](#).

In the **baseline architecture**, those strategies are implemented by unprovisioning and then tearing down the stamp with each update. In this design, complete unprovisioning isn't possible because the platform team owns some resources. So the deployment model was changed.

Deployment model

The **baseline architecture** uses Blue-Green model to deploy application updates. New stamps with changes are deployed alongside existing stamps. After traffic is moved to the new stamp, the existing stamp is destroyed.

In this case, the given peered virtual network is non-ephemeral. The stamp isn't allowed to create the virtual network or peering to the regional hub. You'll need to reuse those resources in each deployment.

Canary model can achieve safe deployment with the option to roll back. First, a new stamp is deployed with code changes. The deployment pipeline references the pre-

provisioned virtual network and allocates subnets, deploys resources, adds private endpoints. Then, it shifts traffic to these subnets in this pre-provisioned virtual network.

When the existing stamp is no longer required, all stamp resources are deleted by the pipeline, except for the platform-owned resources. The virtual network, diagnostic settings, peering, IP address space, DNS configuration, and role-based access control (RBAC) are preserved. At this point, the stamp is in a clean state, and ready for the next new deployment.

DINE (deploy-if-not-exists) Azure policies

Azure application landing zones might have DINE (deploy-if-not-exists) Azure policies. Those checks ensure that deployed resources meet corporate standards in application landing zones, even when they're owned by the application team. There might be a mismatch between your deployment and the final resource configuration.

Understand the impact of all DINE policies that will be applied to your resources. If there are changes to resource configuration, incorporate the intention of the policies into your declarative deployments early in the workload's development cycle. Otherwise, there might be a drift leading to delta between the desired state and the deployed state.

Deployment subscription access management

Treat subscription boundaries as your security boundaries to limit the blast radius. Threats can impact the workload's reliability.

Platform team

- Give the application teams authorization for operations with permissions scoped to the application landing zone subscription.

If you're running multiple deployments within a single subscription, a breach will impact both deployments. Running deployments in dedicated subscriptions is recommended. Create service principals per environment for maintaining logical separation.

The service principal should provide autonomy over workload resources. Also, it should have restrictions in place that will prevent excessive manipulation of the platform resources within the subscription.

Monitoring considerations

The Azure landing zone platform provides shared observability resources as part of the Management subscriptions. The centralized operations team [encourage the application teams to use the federated model](#). But for mission-critical workloads, a single, centralized observability store isn't recommended because it can potentially be a single point of failure. Mission-critical workloads also generate telemetry that might not be applicable or actionable for centralized operations teams.

So, an autonomous approach for monitoring is highly recommended. Workload operators are ultimately responsible for the monitoring and must have access to all data that represents overall health.

The **baseline architecture** follows that approach and is continued in this reference architecture. Azure Log Analytics and Azure Application Insights are deployed regionally and globally to monitor resources in those scopes. Aggregating logs, creating dashboards, and alerting is in scope for your team. Take advantage of Azure Diagnostics capabilities that send metrics and logs to various sinks to support platform requirements for log & metric collection.

Health model

Mission-critical design methodology requires a system [health model](#). When you're defining an overall health score, include in-scope platform landing zone flows that the application depends on. Build log, health, and alert queries to perform cross-workspace monitoring.

Platform team

- Grant role-based access control (RBAC) query and read log sinks for relevant platform resources that are used in the critical path of the mission-critical application.
- Support the organizational goal of reliability toward the mission-critical workload by giving the application team enough permission to do their operations.

In this architecture, the health model includes logs and metrics from resources provisioned in Connectivity subscription. If you extend this design to reach an on-premises resource such as a database, the health model must include network connectivity to that resource, including security boundaries like network virtual appliances in Azure *and* on-premises. This information is important to quickly determine the root cause and remediate the reliability impact. For example, did the failure occur when trying to route to the database, or was there an issue with the database?

Refer to: [Well-architected mission-critical workloads: Health modeling](#).

Integration with the platform-provided policies and rules

The application landing zone subscription inherits Azure policies, Azure Network Manager rules, and other controls from its management group. The platform team provides those guardrails.

For deployments, don't depend on the platform-provided policies exclusively, because:

- They aren't designed to cover the needs of individual workloads.
- The policies and rules might get updated or removed outside your team, and so can impact reliability.

It's highly recommended that you create and assign Azure policies within your deployments. This effort might lead to some duplication but that's acceptable, considering the potential impact on reliability of the system. If there are changes in the platform policies, the workload policies will still be in effect locally.

Platform team

- Involve the application team in the change management process of policies as they evolve.
- Be aware of the policies used by the application team. Evaluate if they should be added to the management group.

Deploy this architecture

The networking aspects of this architecture are implemented in the Mission-critical Connected implementation.

Implementation: Mission-critical Connected

ⓘ Note

The Connected implementation is intended to illustrate a mission-critical workload that relies on organizational resources, integrates with other workloads, and uses shared services. The implementation assumes that a Connectivity subscription already exists.

Next steps

Review the networking and connectivity design area in Azure Well-architected Framework.

Design area: Networking and connectivity

Related resources

In addition to the [Azure services used in the baseline architecture](#), these services are important for this architecture.

- [Azure Management Groups](#)
- [Azure Policy](#)
- [Azure Network Manager](#)
- [Azure Monitor](#)
- [Virtual Networks](#)
- [Route tables](#)

Mission-critical baseline with App Service

Azure App Service

Azure Front Door

Azure Cache for Redis

Azure App Configuration

Azure Monitor

This article describes how to deploy mission-critical web applications by using Azure App Service. The architecture uses the [reliable web app pattern](#) as a starting point. Use this architecture if you have a legacy workload and want to adopt platform-as-a-service (PaaS) services.

[Reliable web app pattern for .NET](#) provides guidance for updating or replatforming web apps that you move to the cloud, minimizing required code changes, and targeting a service-level objective (SLO) of 99.9%. Mission-critical workloads have high reliability and availability requirements. To reach an SLO of 99.95%, 99.99%, or higher, you need to apply supplemental mission-critical design patterns and operational rigor. This article describes key technical areas and how to implement and introduce mission-critical design practices.

ⓘ Note

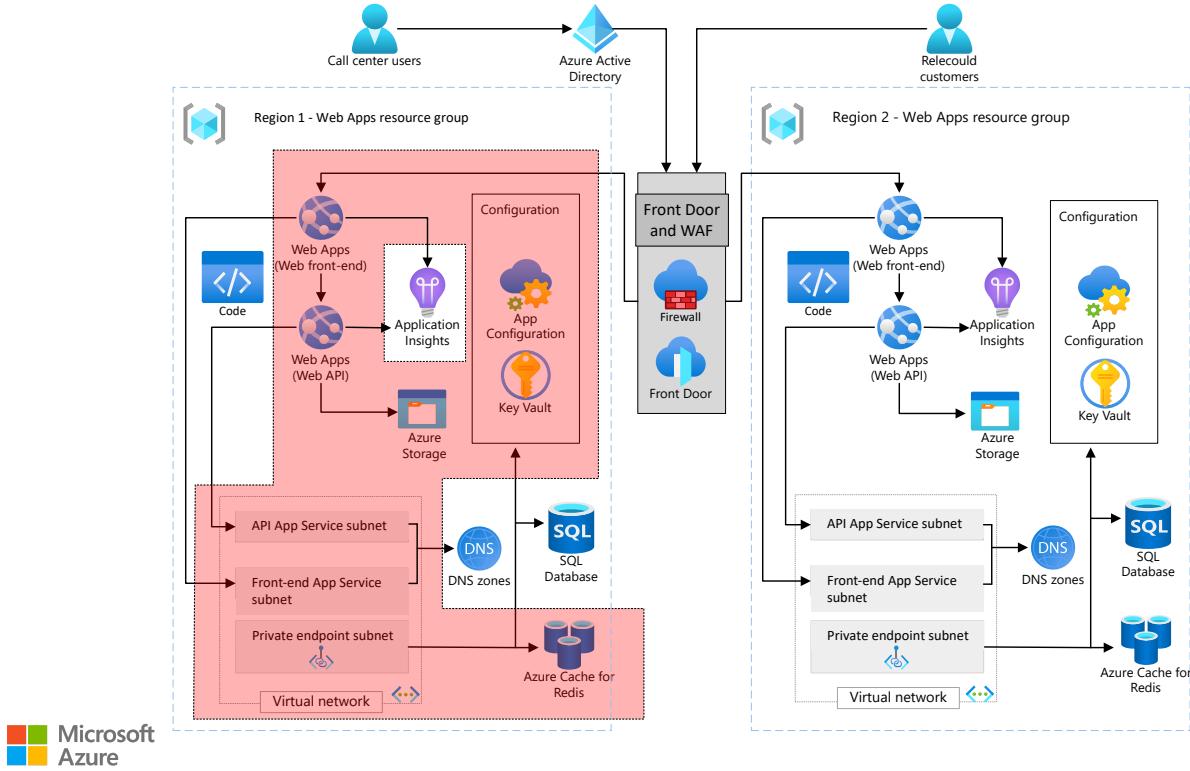
The guidance in this article is based on the design methodology and best practices in the [Well-Architected Framework mission-critical workload series](#).

The following sections describe how to:

- Review the existing workload to understand its components, user and system flows, and availability and scalability requirements.
- Develop and implement a [scale-unit architecture](#) to optimize end-to-end scalability through compartmentalization and to standardize the process of adding and removing capacity.
- Implement stateless, ephemeral scale units or deployment stamps to enable scalability and zero-downtime deployments.
- Determine if you can split the workload into components to prepare for scalability. Individual components are required for scalability and decoupling flows.
- Prepare for [global distribution](#) by deploying a workload across more than one Azure region to improve proximity to the customer and prepare for potential regional outages.
- Decouple components and implement an event-driven architecture.

Architecture

The following diagram applies the previous considerations to the [reliable web app pattern](#).



Download a [Visio file](#) of this architecture.

The red box represents a scale unit with services that scale together. To effectively scale them together, optimize each service's size, SKU, and available IP addresses. For example, the maximum number of requests that Azure App Configuration serves correlates to the number of requests per second that a scale unit provides. When you add more capacity in a region, you must also add more individual scale units.

These individual scale units don't have any inter-dependencies and only communicate with shared services outside of the individual scale unit. You can test independent scale units upfront. To avoid affecting other areas of deployment, roll out independent scale units and introduce the option to replace services in a new release.

For mission-critical workloads, independent scale units are temporary, which optimizes the rollout processes and provides scalability within and across regions. Avoid storing state in independent scale units. Consider using Azure Cache for Redis for storage in the scale unit, and only store critical state or data that's also stored in the database. If there's a scale-unit outage or you switch to another scale unit, there might be a slowdown or a new sign-in required, but Azure Cache for Redis still runs.

Application Insights is excluded from the scale unit. Exclude services that store or monitor data. Separate them into their own resource group with their own lifecycle.

When you replace a scale unit or deploy a new one, keep historical data and use one instance per region.

For more information, see [Application design of mission-critical workloads on Azure](#).

Components

This architecture uses the following components.

- [App Service](#) is the application-hosting platform.
- [Azure Cache for Redis](#) caches requests.
- [App Configuration](#) stores configuration settings.
- [Azure SQL](#) is the back-end database.
- [Application Insights](#) gets telemetry from the application.

Alternatives

In the reliable web app pattern, you can:

- Use Azure Kubernetes Service (AKS) instead of App Service. This option works well for complex workloads that have a large number of microservices. AKS provides more control over the underlying infrastructure and allows complex multitier setups.
- Containerize the workload. App Service supports containerization, but in this example the workload isn't containerized. Use containers to increase reliability and portability.

For more information, see [Application platform considerations for mission-critical workloads on Azure](#).

Choose the application platform

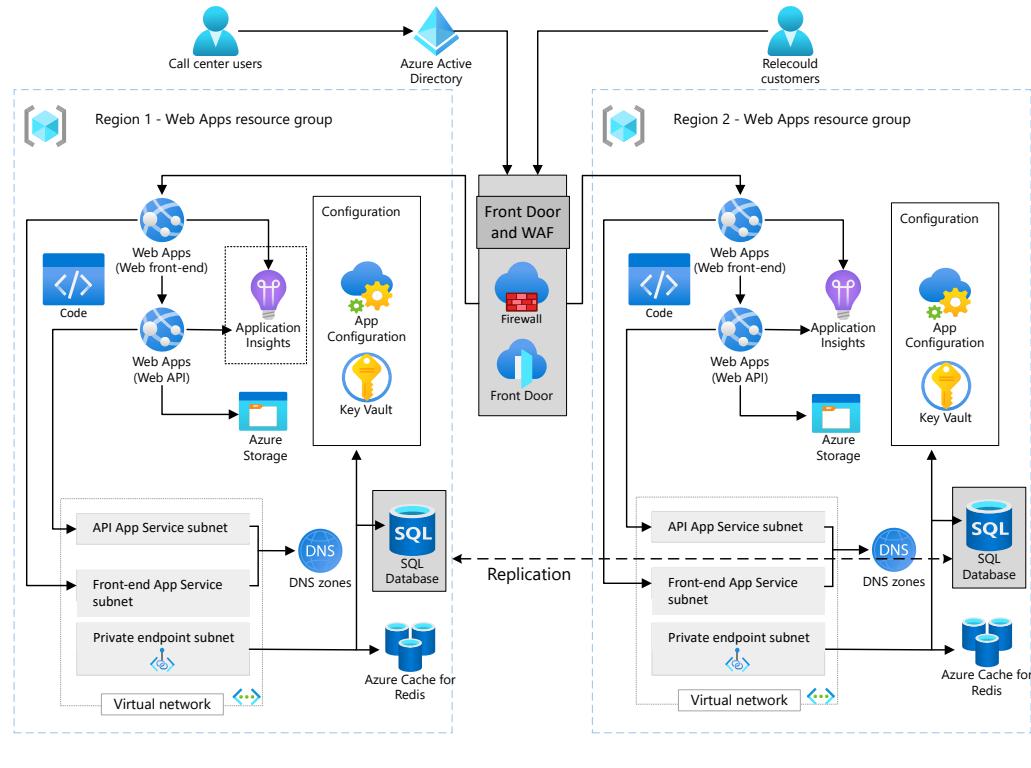
The level of availability depends on your choice and configuration of the application platform. Consider the following mission-critical guidance:

- Use availability zones when possible.
- Select the right platform service for your workload.
- Containerize the workload.

Availability sets spread deployments across multiple fault and update domains within a datacenter. *Availability zones* spread deployments across individual datacenters within an Azure region. Availability zones are often prioritized, but which strategy you use depends on your workload. For example, latency-sensitive or chatty workloads might benefit from prioritizing availability sets. If you spread the workload across availability zones, it can increase latency and cost for cross-zone traffic. When you use availability zones, ensure that all services in a scale unit support them. All services in the reliable web app pattern support availability zones.

Choose the data platform

The database platform you choose affects the overall workload architecture, especially the platform's active-active or active-passive configuration support. The reliable web app pattern uses Azure SQL, which doesn't natively support active-active deployments with write operations in more than one instance. So the database level is limited to an active-passive strategy. An active-active strategy on the application level is possible if there are read-only replicas and you write to a single region only.



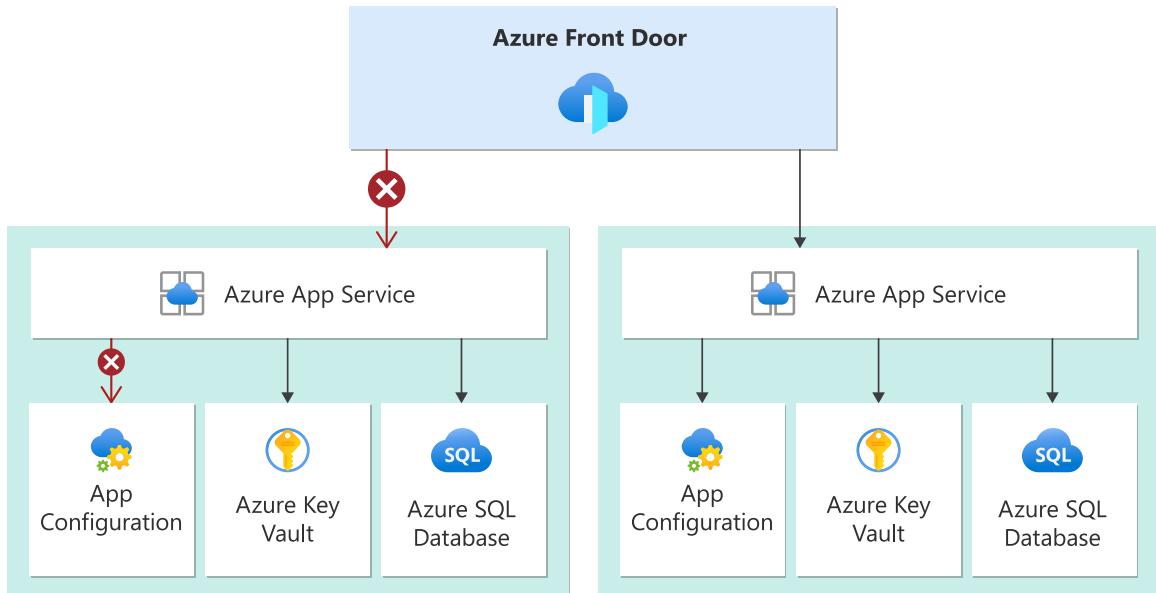
Download a [Visio file](#) of this architecture.

Multiple databases are common in complex architectures, such as microservices architectures that have a database for each service. Multiple databases allow the adoption of a multi-primary write database like Azure Cosmos DB, which improves high availability and low latency. Cross-region latency can create limitations. It's crucial to

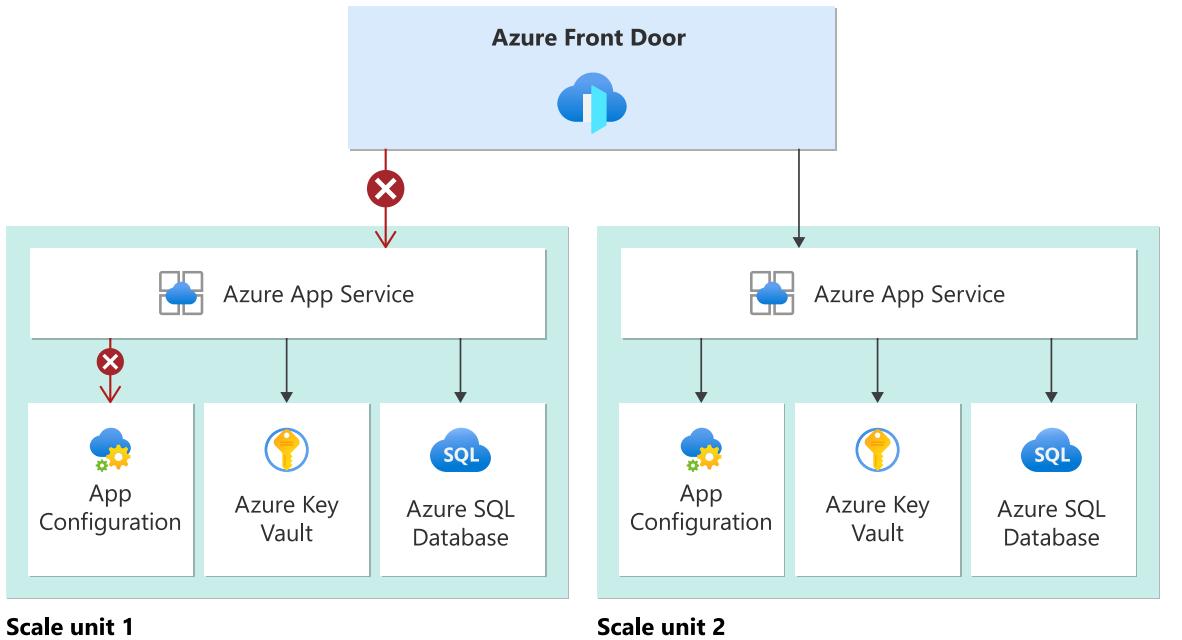
consider nonfunctional requirements and factors like consistency, operability, cost, and complexity. Enable individual services to use separate data stores and specialized data technologies to meet their unique requirements. For more information, see [Data platform considerations for mission-critical workloads on Azure](#).

Define a health model

In complex multtier workloads that spread across multiple datacenters and geographical regions, you must define a health model. Define user and system flows, specify and understand the dependencies between the services, understand the effect that outages or a performance degradation on one of the services can have on the overall workload, and monitor and visualize the customer experience to enable proper monitoring and improve manual and automated actions.



The previous diagram shows how an outage or a degradation of a single component, like App Configuration, can cause potential performance degradation for the customer.



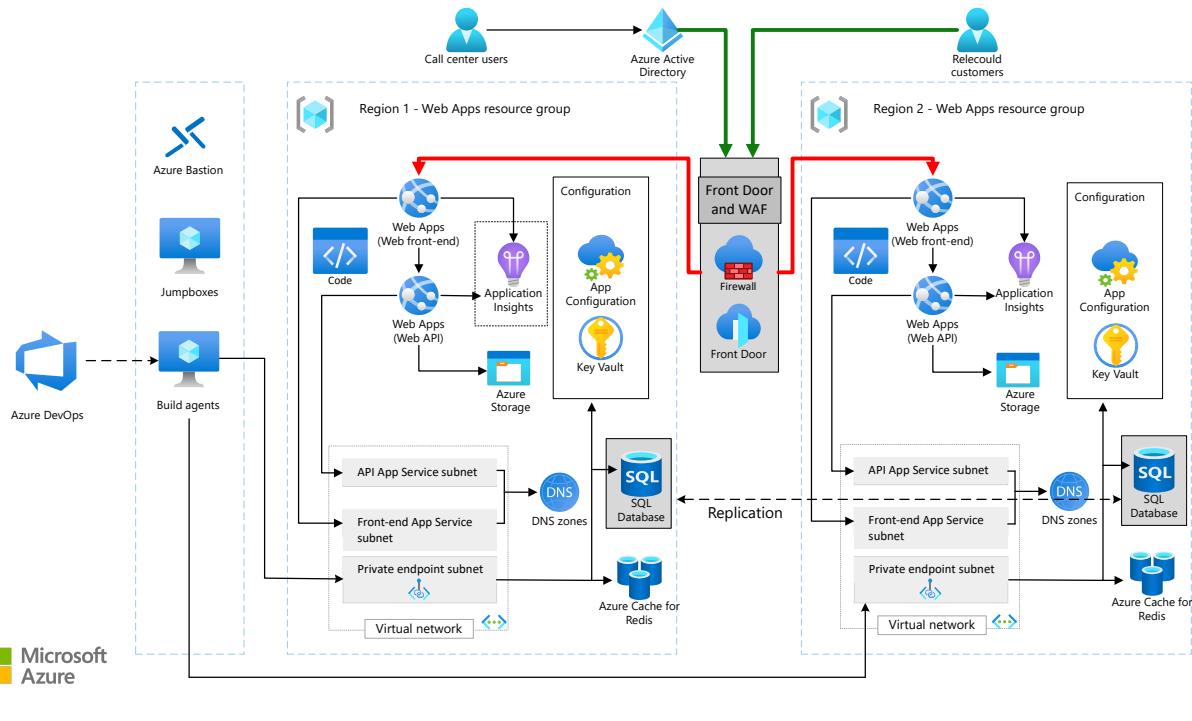
When you separate components into scale units, it allows you to stop sending traffic to the affected part of the application, such as an affected scale unit or the complete region.

For more information, see [Health modeling and observability of mission-critical workloads on Azure](#).

Security and networking

There are strict networking and security requirements for workloads that migrate from an on-premises enterprise deployment. Not all established on-premises processes translate into a cloud environment. Evaluate these requirements if they're applicable in cloud environments.

Identity is often the primary security perimeter for cloud-native patterns. Enterprise customers might need more substantial security measures. To address their network security requirements, most of the Azure PaaS services can use Azure Private Link to implement the network as a security perimeter. Private Link can ensure that services are only accessible from within a virtual network. All services are accessible via private endpoints only. The following diagram shows how the only public internet-facing endpoint is Azure Front Door.



[Download a Visio file](#) of this architecture.

For the reliable web app pattern to set up a network as a security perimeter, it must use:

- Private Link for all services that support it.
- Azure Front Door premium as the only internet-facing public endpoint.
- Jumpboxes to access services via Azure Bastion.
- Self-hosted build agents that can access the services.

Another common network requirement for mission-critical applications is to restrict egress traffic to prevent data exfiltration. Restrict egress traffic by routing an Azure firewall through a proper firewall device and filtering it with the device. The [Azure mission-critical baseline architecture with network controls](#) uses a firewall and Private Link.

Deployment and testing

Downtime caused by erroneous releases or human error can be an issue for a workload that needs to always be available. Here are some key areas to consider:

- Zero-downtime deployments
- Ephemeral blue/green deployments
- Analyzing the lifecycle of individual components and grouping them together
- Continuous validation

[Zero-downtime deployments](#) are key for mission-critical workloads. A workload that needs to always be up and running can't have a maintenance window to roll out newer

versions. To work around this limitation, the Azure mission-critical architecture follows the zero-downtime deployments pattern. Changes are rolled out as new scale units or stamps that are tested end to end before traffic is incrementally routed to them. After all traffic is routed to the new stamp, old stamps are disabled and removed.

For more information, see [Deployment and testing for mission-critical workloads on Azure](#).

Next steps

- Learning path: Build mission-critical workloads on Azure
- Challenge project: Design a mission-critical web application
- Learn module: Design a health model for your mission-critical workload

Related resources

- Mission-critical baseline architecture on Azure
- Mission-critical baseline architecture with network controls
- Mission-critical baseline architecture in an Azure landing zone
- Continuous validation with Azure Load Testing and Azure Chaos Studio

Application platform considerations for mission-critical workloads

Article • 01/27/2023

A key design area of any mission critical architecture is the application platform. Platform refers to the infrastructure components and Azure services that must be provisioned to support the application. Here are some overarching recommendations.

- Design in layers. Choose the right set of services, their configuration, and the application-specific dependencies. This layered approach helps in creating **logical and physical segmentation**. It's useful in defining roles and functions, and assigning appropriate privileges, and deployment strategies. This approach ultimately increases the reliability of the system.
- A mission-critical application must be highly reliable and resistant to datacenter and regional failures. Building **zonal and regional redundancy** in an active-active configuration is the main strategy. As you choose Azure services for your application's platform, consider their Availability Zones support and deployment and operational patterns to use multiple Azure regions.
- Use a *scale units*-based architecture to handle increased load. Scale units allow you to logically group resources and a unit can be **scaled independent of other units** or services in the architecture. Use your capacity model and expected performance to define the boundaries of, number of, and the baseline scale of each unit.

In this architecture, the application platform consists of global, deployment stamp, and regional resources. The regional resources are provisioned as part of a deployment stamp. Each stamp equates to a scale unit and, in case it becomes unhealthy, can be entirely replaced.

The resources in each layer have distinct characteristics. For more information, see [Architecture pattern of a typical mission-critical workload](#).

Characteristics	Considerations
Lifetime	What is the expected lifetime of 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?

Characteristics	Considerations
Reach	Is the resource required to be globally distributed? Can the resource communicate with other resources, globally or in regions?
Dependencies	What's the dependency on other resources, globally or in other regions?
Scale limits	What is the expected throughput for that resource at that layer? How much scale is provided by the resource to fit that demand?
Availability/disaster recovery	What's the impact on availability or disaster at this layer? Would it cause a systemic outage or only localized capacity or availability issue?

Global resources

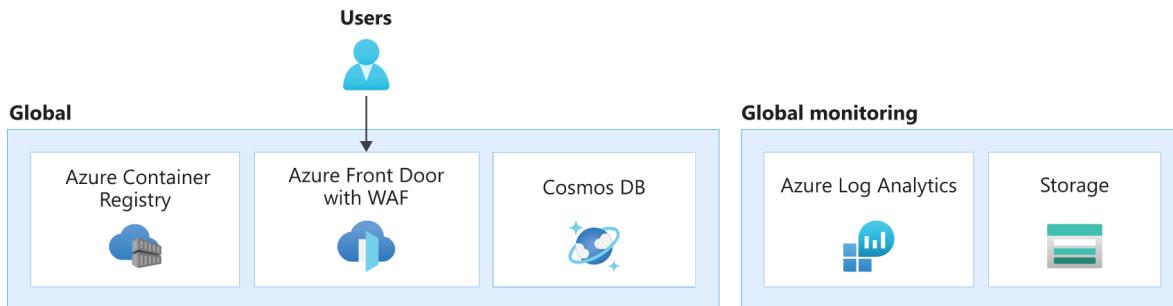
Certain resources in this architecture are shared by resources deployed in regions. In this architecture, they are used to distribute traffic across multiple regions, store permanent state for the whole application, and cache global static data.

Characteristics	Layer Considerations
Lifetime	These resources are expected to be long living. 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.
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. 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 of 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 as such they should be able to scale such that they can handle throughput of the system as a whole.
Availability/disaster recovery	Because regional and stamp resources can consume global resources or are fronted by them, it's critical that global resources are configured with high availability and disaster recovery for the health of the whole system.

In this architecture, global layer resources are [Azure Front Door](#), [Azure Cosmos DB](#), [Azure Container Registry](#), and [Azure Log Analytics](#) for storing logs and metrics from

other global layer resources.

There are other foundational resources in this design, such as Azure Active Directory (AD) and Azure DNS. They have been omitted in this image for brevity.



Global load balancer

Azure Front Door is used as the *only entry point* for user traffic. Azure guarantees that Azure Front Door will deliver the requested content without error 99.99% of the time. For more details, see [Front Door service limits](#). If Front Door becomes unavailable, the end user will see the system as being down.

The Front Door instance sends traffic to the configured backend services, such as the compute cluster that hosts the API and the frontend SPA. **Backend misconfigurations in Front Door can lead to outages.** To avoid outages due to misconfigurations, you should extensively test your Front Door settings.

Another common error can come from **misconfigured or missing TLS certificates**, which can prevent users from using the front end or Front Door communicating to the backend. Mitigation might require manual intervention. For example, you might choose to roll back to the previous configuration and re-issue the certificate, if possible. Regardless, expect unavailability while changes take effect. Using managed certificates offered by Front door is recommended to reduce the operational overhead, such as handling expiration.

Front Door offers many additional capabilities besides global traffic routing. An important capability is the Web Application Firewall (WAF), because Front Door is able to inspect traffic which is passing through. When configured in the *Prevention* mode, it will block suspicious traffic before even reaching any of the backends.

For information about Front Door capabilities, see [Frequently asked questions for Azure Front Door](#).

For other considerations about global distribution of traffic, see [Misson-critical guidance in Well-architected Framework: Global routing](#).

Container Registry

Azure Container Registry is used to store Open Container Initiative (OCI) artifacts, specifically helm charts and container images. It doesn't participate in the request flow and is only accessed periodically. Container registry is required to exist before stamp resources are deployed and shouldn't have dependency on regional layer resources.

Enable zone redundancy and geo-replication of registries so that runtime access to images is fast and resilient to failures. In case of unavailability, the instance can then fail over to replica regions and requests are automatically re-routed to another region. Expect transient errors in pulling images until failover is complete.

Failures can also occur if images are deleted inadvertently, new compute nodes won't be able to pull images, but existing nodes can still use cached images. The primary **strategy for disaster recovery is redeployment**. The artifacts in a container registry can be regenerated from pipelines. Container registry must be able to withstand many concurrent connections to support all of your deployments.

It's recommended that you use the Premium SKU to enable geo replication. The zone redundancy feature ensures resiliency and high availability within a specific region. In case of a regional outage, replicas in other regions are still available for data plane operations. With this SKU you can restrict access to images through private endpoints.

For more details, see [Best practices for Azure Container Registry](#).

Database

It's recommended that all state is stored globally in a database separated from regional stamps. Build redundancy by deploying the database across regions. For mission-critical workloads, **synchronizing data across regions should be the primary concern**. Also, in case of a failure, write requests to the database should still be functional.

Data replication in an active-active configuration is strongly recommended. The application should be able to instantly connect with another region. All instances should be able to handle read *and* write requests.

For more information, see [Data platform for mission-critical workloads](#).

Global monitoring

Azure Log Analytics is used to store diagnostic logs from all global resources. It's recommended that you restrict daily quota on storage especially on environments that

are used for load testing. Also, set retention policy. These restrictions will prevent any overspend that is incurred by storing data that isn't needed beyond a limit.

Considerations for foundational services

The system is likely to use other critical platform services that can cause the entire system to be at risk, such as Azure DNS and Azure Active Directory (AD). Azure DNS guarantees 100% availability SLA for valid DNS requests. Azure Active Directory guarantees at least 99.99% uptime. Still, you should be aware of the impact in the event of a failure.

Taking hard dependency on foundational services is inevitable because many Azure services depend on them. Expect disruption in the system if they are unavailable. For instance:

- Azure Front Door uses Azure DNS to reach the backend and other global services.
- Azure Container Registry uses Azure DNS to fail over requests to another region.

In both cases, both Azure services will be impacted if Azure DNS is unavailable. Name resolution for user requests from Front Door will fail; Docker images won't be pulled from the registry. Using an external DNS service as backup won't mitigate the risk because many Azure services don't allow such configuration and rely on internal DNS. Expect full outage.

Similarly, Azure AD is used for control plane operations such as creating new AKS nodes, pulling images from Container Registry, or accessing Key Vault on pod startup. If Azure AD is unavailable, existing components shouldn't be affected, but overall performance may be degraded. New pods or AKS nodes won't be functional. So, in case scale out operations are required during this time, expect decreased user experience.

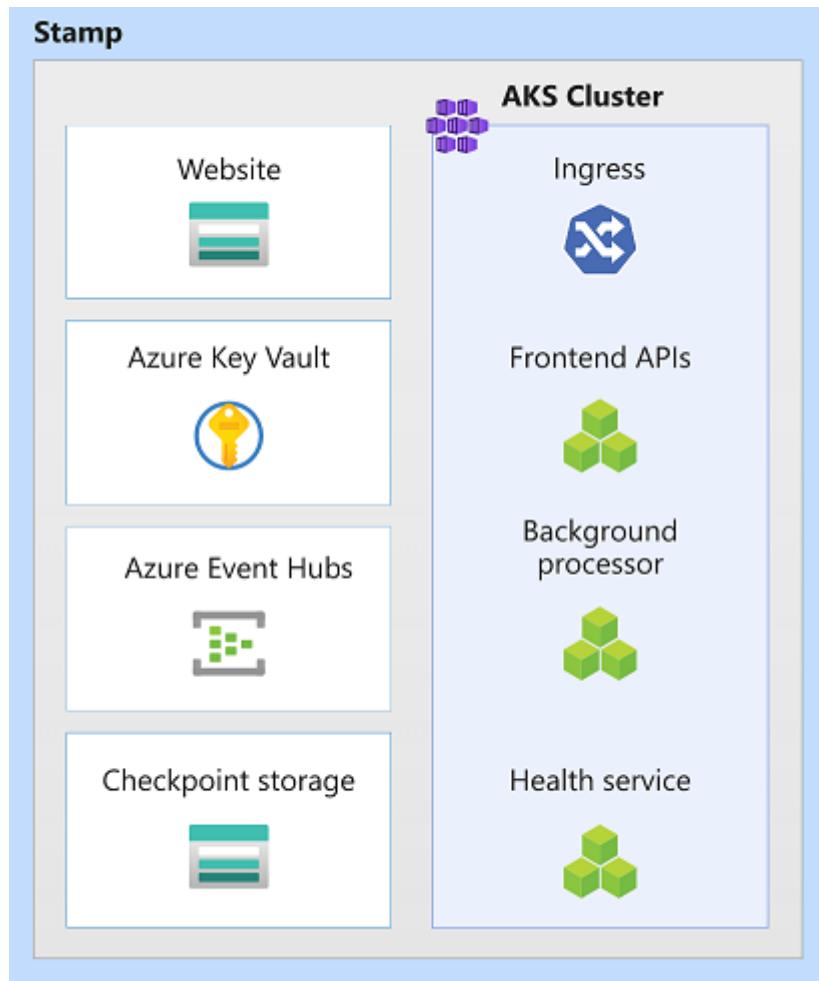
Regional deployment stamp resources

In this architecture, the deployment stamp deploys the workload and provisions 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.

Characteristics	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.

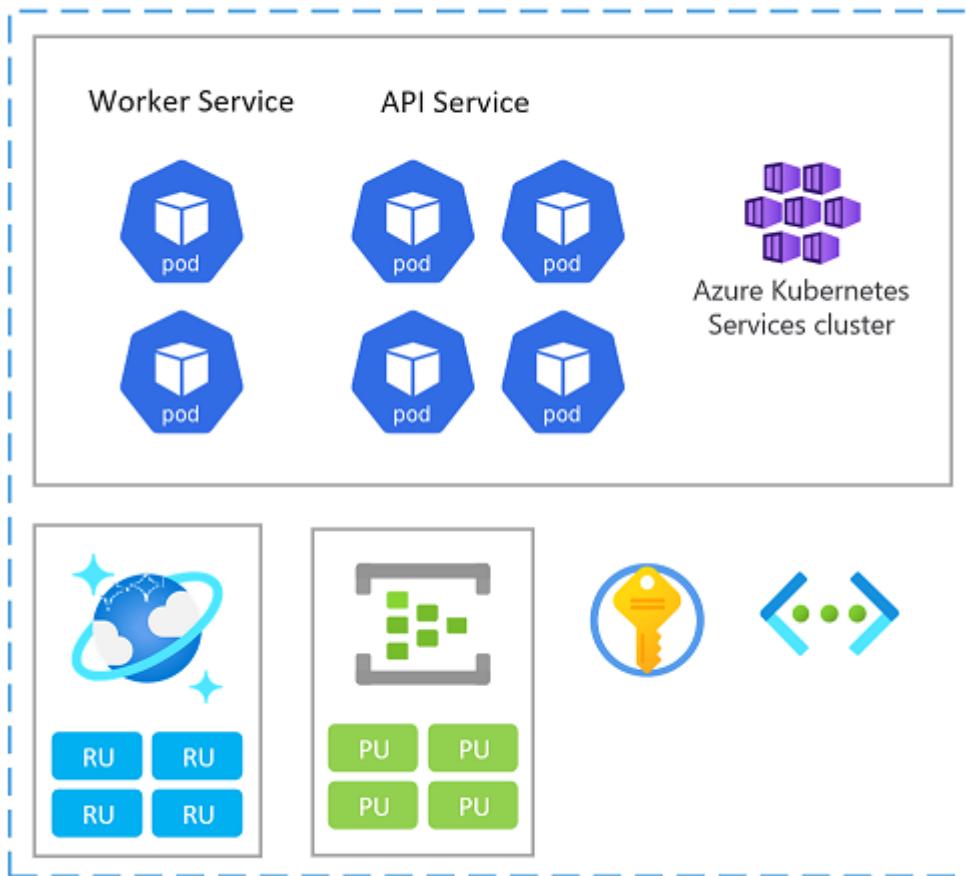
Characteristics	Considerations
State	Because stamps are ephemeral and can be destroyed at any time, a stamp should be stateless as much as possible.
Reach	Can communicate with regional and global resources. However, communication with other regions or other stamps should be avoided. In this architecture, there isn't a need for these resources to be globally distributed.
Dependencies	<p>The stamp resources must be independent. That is, they shouldn't rely on other stamps or components in other regions. They are expected to have regional and global dependencies.</p> <p>The main shared component is the database layer and container registry. This component requires synchronization at runtime.</p>
Scale limits	Throughput is established through testing. The throughput of the overall stamp is limited to the least performant resource. Stamp throughput needs to take into account the estimated high-level of demand and any failover as the result of another stamp in the region becoming unavailable.
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.

In this architecture, stamp resources are [Azure Kubernetes Service](#), [Azure Event Hubs](#), [Azure Key Vault](#), and [Azure Blob Storage](#).



Scale unit

A stamp can also be considered as a scale unit (SU). All components and services within a given stamp are configured and tested to serve requests in a given range. Here's an example of a scale unit used in the implementation.



Each scale unit is deployed into an Azure region and is therefore primarily handling traffic from that given area (although it can take over traffic from other regions when needed). This geographic spread will likely result in load patterns and business hours that might vary from region to region and as such, every SU is designed to scale-in/-down when idle.

You can deploy a new stamp to scale. Inside a stamp, individual resources can also be [units of scale](#).

Here are some scaling and availability considerations when choosing Azure services in a unit:

- **Evaluate capacity relations** between all resources in a scale unit. For example, to handle 100 incoming requests, 5 ingress controller pods and 3 catalog service pods and 1000 RUs in Azure Cosmos DB would be needed. So, when autoscaling the ingress pods, expect scaling of the catalog service and Azure Cosmos DB RUs given those ranges.
- **Load test the services** to determine a range within which requests will be served. Based on the results configure minimum and maximum instances and target metrics. When the target is reached, you can choose to automate scaling of the entire unit.

- Review the Azure subscription scale limits and quotas to support the capacity and cost model set by the business requirements. Also check the limits of individual services in consideration. Because units are typically deployed together, factor in the subscription resource limits that are required for canary deployments. For more information, see [Azure service limits](#).
- Choose services that support availability zones to build redundancy. This might limit your technology choices. See [Availability Zones](#) for details.

For other considerations about the size of a unit, and combination of resources, see [Mission-critical guidance in Well-architected Framework: Scale-unit architecture](#).

Compute cluster

To containerize the workload, each stamp needs to run a compute cluster. In this architecture, Azure Kubernetes Service (AKS) is chosen because Kubernetes is the most popular compute platform for modern, containerized applications.

The lifetime of the AKS cluster is bound to the ephemeral nature of the stamp. **The cluster is stateless** and doesn't have persistent volumes. It uses ephemeral OS disks instead of managed disks because they aren't expected to receive application or system-level maintenance.

To increase reliability, the cluster is configured to **use all three availability zones** in a given region. This makes it possible for the cluster to use [AKS Uptime SLA](#) that guarantees 99.95% SLA availability of the AKS control plane.

Other factors such as scale limits, compute capacity, subscription quota can also impact reliability. If there isn't enough capacity or limits are reached, scale out and scale up operations will fail but existing compute is expected to function.

The cluster has autoscaling enabled to let node pools **automatically scale out if needed**, which improves reliability. When using multiple node pools, all node pools should be autoscaled.

At the pod level, the Horizontal Pod Autoscaler (HPA) scales pods based on configured CPU, memory, or custom metrics. Load test the components of the workload to establish a baseline for the autoscaler and HPA values.

The cluster is also configured for **automatic node image upgrades** and to scale appropriately during those upgrades. This scaling allows for zero downtime while upgrades are being performed. If the cluster in one stamp fails during an upgrade, other clusters in other stamps shouldn't be affected, but upgrades across stamps should occur

at different times to maintain availability. Also, cluster upgrades are automatically rolled across the nodes so that they aren't unavailable at the same time.

Some components such as cert-manager and ingress-nginx require container images from external container registries. If those repositories or images are unavailable, new instances on new nodes (where the image isn't cached) might not be able to start. This risk could be mitigated by importing these images to the environment's Azure Container Registry.

Observability is critical in this architecture because stamps are ephemeral. Diagnostic settings are configured to store all log and metric data in a regional Log Analytics workspace. Also, AKS Container Insights is enabled through an in-cluster OMS Agent. This agent allows the cluster to send monitoring data to the Log Analytics workspace.

For other considerations about the compute cluster, see [Misson-critical guidance in Well-architected Framework: Container Orchestration and Kubernetes](#).

Key Vault

Azure Key Vault is used to store global secrets such as connection strings to the database and stamp secrets such as the Event Hubs connection string.

This architecture uses a [Secrets Store CSI driver](#) in the compute cluster to get secrets from Key Vault. Secrets are needed when new pods are spawned. If Key Vault is unavailable, new pods might not get started. As a result, there might be disruption; scale out operations can be impacted, updates can fail, new deployments can't be executed.

Key Vault has a limit on the number of operations. Due to the automatic update of secrets, the limit can be reached if there are many pods. You can **choose to decrease the frequency of updates** to avoid this situation.

For other considerations on secret management, see [Misson-critical guidance in Well-architected Framework: Data integrity protection](#).

Event Hubs

The only stateful service in the stamp is the message broker, Azure Event Hubs, which stores requests for a short period. The broker serves the **need for buffering and reliable messaging**. The processed requests are persisted in the global database.

In this architecture, Standard SKU is used and zone redundancy is enabled for high availability.

Event Hubs health is verified by the HealthService component running on the compute cluster. It performs periodic checks against various resources. This is useful in detecting unhealthy conditions. For example, if messages can't be sent to the event hub, the stamp would be unusable for any write operations. HealthService should automatically detect this condition and report unhealthy state to Front Door, which will take the stamp out of rotation.

For scalability, enabling auto-inflate is recommended.

For more information, see [Messaging services for mission-critical workloads](#).

For other considerations about messaging, see [Mission-critical guidance in Well-architected Framework: Asynchronous messaging](#).

Storage accounts

In this architecture two storage accounts are provisioned. Both accounts are deployed in zone-redundant mode (ZRS).

One account is used for Event Hubs checkpointing. If this account isn't responsive, the stamp won't be able to process messages from Event Hubs and might even impact other services in the stamp. This condition is periodically checked by the HealthService, which is one of the application components running in the compute cluster.

The other is used to host the UI single-page application. If serving of the static web site has any issues, Front Door will detect the issue and won't send traffic to this storage account. During this time, Front Door can use cached content.

For more information about recovery, see [Disaster recovery and storage account failover](#).

Regional resources

A system can have resources that are deployed in region but outlive the stamp resources. In this architecture, observability data for stamp resources are stored in regional data stores.

Characteristics	Consideration
Lifetime	The resources share the lifetime of the region and out live the stamp resources.
State	State stored in a region cannot live beyond the lifetime of the region. If state needs to be shared across regions, consider using a global data store.

Characteristics	Consideration
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.

Monitoring data for stamp resources

Deploying monitoring resources is a typical example for regional resources. In this architecture, each region has an individual Log Analytics workspace configured to store all log and metric data emitted from stamp resources. Because regional resources outlive stamp resources, **data is available even when the stamp is deleted**.

Azure Log Analytics and Azure Application Insights are used to store logs and metrics from the platform. It's recommended that you restrict daily quota on storage especially on environments that are used for load testing. Also, set retention policy to store all data. These restrictions will prevent any overspend that is incurred by storing data that isn't needed beyond a limit.

Similarly, Application Insights is also deployed as a regional resource to collect all application monitoring data.

For design recommendations about monitoring, see [Mission-critical guidance in Well-architected Framework: Health modeling](#).

Next steps

Deploy the reference implementation to get a full understanding of the resources and their configuration used in this architecture.

[Implementation: Mission-Critical Online](#)

Application design considerations for mission-critical workloads

Article • 11/04/2022

The [baseline mission critical reference architecture](#) illustrates a highly reliable workload through a simple online catalog application. The end users can browse through a catalog of items, see details of an item, and post ratings and comments for items. This article focuses on reliability and resiliency aspects of a mission-critical application, such as asynchronous processing of requests and how to achieve high throughput within a solution.

ⓘ Important



The guidance is backed by a production-grade [reference implementation](#) which showcases mission critical application development on Azure. This implementation can be used as a basis for further solution development in your first step towards production.

Application composition

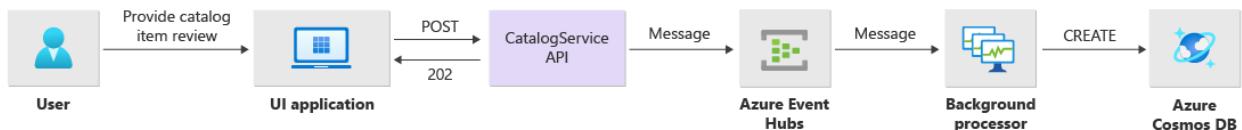
For high-scale mission critical applications, it's essential to **optimize the architecture for end-to-end scalability and resilience**. This state can be achieved through separation of components into functional units that can operate independently. Apply this separation at all levels on the application stack, allowing each part of the system to scale independently and meet changes in demand.

An example of that approach is shown in the implementation. The application uses stateless API endpoints, which decouple long-running write requests asynchronously through a messaging broker. The workload is composed in a way that the whole AKS cluster and other dependencies in the stamp can be deleted and recreated at any time. The main components are:

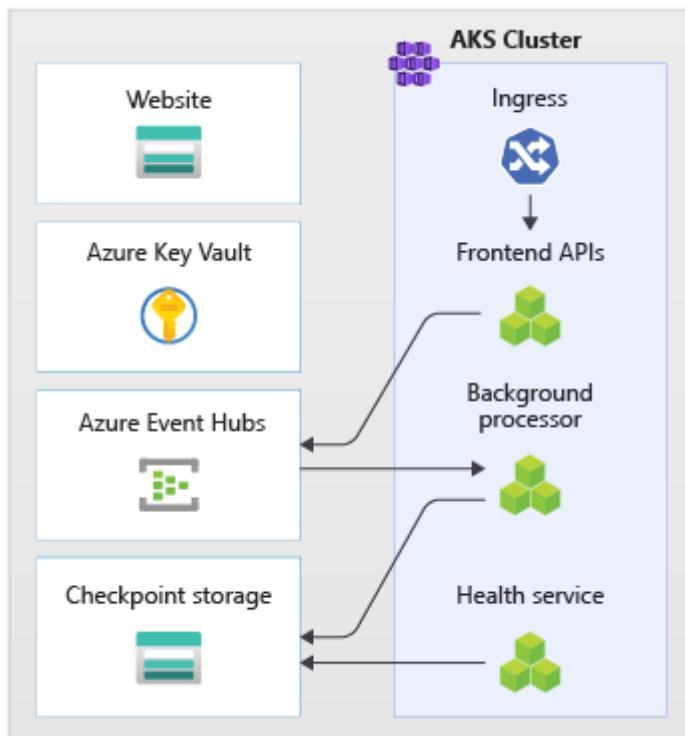
- **User interface (UI)**: single-page web application accessed by end users is hosted in Azure Storage Account's static website hosting.
- **API (CatalogService)**: REST API called by the UI application, but available for other potential client applications.
- **Worker (BackgroundProcessor)**: background worker, which processes write requests to the database by listening to new events on the message bus. This component

does not expose any APIs.

- **Health service API** (`HealthService`): used to report the health of the application by checking if critical components (database, messaging bus) are working.



The API, worker, and health check applications are referred to as **workload** and hosted as containers in a dedicated AKS namespace (called `workload`). There's **no direct communication** between the pods. The pods are **stateless** and able to **scale independently**.



There are other supporting components running in the cluster:

1. **Ingress controller:** Nginx Ingress Controller is used to route incoming requests to the workload and load balance between pods. It is exposed through Azure Load Balancer with a public IP address (but only accessed through Azure Front Door).
2. **Cert manager:** Jetstack's `cert-manager` is used to auto-provision SSL/TLS certificates (using Let's Encrypt) for the ingress rules.
3. **CSI secrets driver:** Azure Key Vault Provider for Secrets Store CSI is used to securely read secrets such as connection strings from Azure Key Vault.
4. **Monitoring agent:** The default OMSAgent configuration is adjusted to reduce the amount of monitoring data sent to the Log Analytics workspace.

Database connection

Due to the ephemeral nature of deployment stamps, avoid persisting state within the stamp as much as possible. State should be persisted in an externalized data store. To support the reliability SLO, that data store needs to be resilient. It's recommended that you use managed (PaaS) services combined with native SDK libraries that automatically handle timeouts, disconnects and other failure states.

In the reference implementation, **Azure Cosmos DB** serves as the main data store for the application. [Azure Cosmos DB](#) was chosen because it provides **multi-region writes**. Each stamp can write to the Azure Cosmos DB replica in the same region with Azure Cosmos DB internally handling data replication and synchronization between regions. **Azure Cosmos DB for NoSQL** is used because it supports all capabilities of the database engine.

For more information, see [Data platform for mission-critical workloads](#).

ⓘ Note

New applications should use Azure Cosmos DB for NoSQL. For legacy applications that use another NoSQL protocol, evaluate the migration path to Azure Cosmos DB.

💡 Tip

For mission-critical applications that prioritize availability over performance, **single-region write and multi-region read** with *Strong consistency* level are recommended.

In this architecture, there's a need to store state temporarily in the stamp for Event Hubs checkpointing. **Azure Storage** is used for that purpose.

All workload components use the Azure Cosmos DB .NET Core SDK to communicate with the database. The SDK includes robust logic to maintain database connections and handle failures. Here are some key configuration settings:

- Uses **Direct connectivity mode**. This is the default setting for .NET SDK v3 because it offers better performance. There are fewer network hops compared to Gateway mode which uses HTTP.
- **Return content response on write** is disabled to prevent the Azure Cosmos DB client from returning the document from Create, Upsert, Patch and Replace

operations to reduce network traffic. Also, this is not needed for further processing on the client.

- **Custom serialization** is used to set the JSON property naming policy to `JsonNamingPolicy.CamelCase` to translate .NET-style properties to standard JSON-style and vice-versa. The default ignore condition ignores properties with null values during serialization (`JsonIgnoreCondition.WhenWritingNull`).
- **Application region** is set to the region of the stamp, which enables the SDK to find the closest connection endpoint (preferably within the same region).

```
C#
```

```
//  
// /src/app/AlwaysOn.Shared/Services/CosmosDbService.cs  
  
CosmosClientBuilder clientBuilder = new  
CosmosClientBuilder(sysConfig.CosmosEndpointUri, sysConfig.CosmosApiKey)  
    .WithConnectionModeDirect()  
    .WithContentResponseOnWrite(false)  
  
.WithRequestTimeout(TimeSpan.FromSeconds(sysConfig.CombosRequestTimeoutSeconds))  
  
.WithThrottlingRetryOptions(TimeSpan.FromSeconds(sysConfig.CombosRetryWaitSeconds), sysConfig.CombosMaxRetryCount)  
    .WithCustomSerializer(new  
CosmosNetSerializer(Globals.JsonSerializerOptions));  
  
if (sysConfig.AzureRegion != "unknown")  
{  
    clientBuilder =  
clientBuilder.WithApplicationRegion(sysConfig.AzureRegion);  
}  
  
_dbClient = clientBuilder.Build();
```

Asynchronous messaging

Loose coupling allows services to be designed in a way that a **service doesn't have dependency on other services**. The *loose* aspect allows a service to operate independently. The *coupling* aspect allows for inter-service communication through well-defined interfaces. In the context of a mission critical application, it facilitates high-availability by preventing downstream failures from cascading to frontends or different deployment stamps.

Key characteristics:

- Services aren't constrained to use the same compute platform, programming language, or operating system.
- Services scale independently.
- Downstream failures don't affect client transactions.
- Transactional integrity is more difficult to maintain, because data creation and persistence happens in separate services. This is also a challenge across messaging and persistence services, as described in [this guidance on idempotent message processing](#).
- End-to-end tracing requires more complex orchestration.

Using well-known design patterns, such as [Queue-Based Load leveling pattern](#) and [Competing Consumers pattern](#), is highly recommended. These patterns help in load distribution from the producer to the consumers and asynchronous processing by consumers. For example, the worker allows the API to accept the request and return to the caller quickly while processing a database write operation separately.

Azure Event Hubs is used as the message broker between the API and worker.

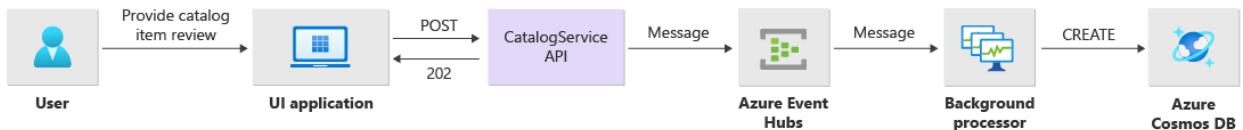
Important

The message broker is not intended to be used as a persistent data store for long periods of time. The Event Hubs service supports [capture feature](#) which allows an event hub to automatically write a copy of messages to a linked Azure Storage account. This keeps utilization in-check but it also serves as a mechanism to backup messages.

Implementation details for write operations

Write operations, such as *post rating* and *post comment* are processed asynchronously. The API first sends a message with all relevant information, such as type of action and comment data, to the message queue and immediately returns `HTTP 202 (Accepted)` with additional `Location` header of the to-be-created object.

Messages in the queue are then processed by `BackgroundProcessor` instances which handle the actual database communication for write operations. `BackgroundProcessor` scales in and out dynamically based on message volume on the queue. The scale out limit of processor instances is defined by the [maximum number of Event Hubs partitions](#) (which is 32 for Basic and Standard tiers, 100 for Premium tier and 1024 for Dedicated tier).



The Azure EventHub Processor library in `BackgroundProcessor` uses Azure Blob Storage to manage partition ownership, load balance between different worker instances, and to track progress using checkpoints. **Writing the checkpoints to the blob storage does not occur after every event** because this would add a prohibitively expensive delay for every message. Instead, the checkpoint writing occurs on a timer-loop (configurable duration with a current setting of 10 seconds):

C#

```

while (!stoppingToken.IsCancellationRequested)
{
    await Task.Delay(TimeSpan.FromSeconds(_sysConfig.BackendCheckpointLoopSeconds),
stoppingToken);
    if (!stoppingToken.IsCancellationRequested && !checkpointEvents.IsEmpty)
    {
        string lastPartition = null;
        try
        {
            foreach (var partition in checkpointEvents.Keys)
            {
                lastPartition = partition;
                if (checkpointEvents.TryRemove(partition, out ProcessEventArgs lastProcessEventArgs))
                {
                    if (lastProcessEventArgs.HasEvent)
                    {
                        _logger.LogDebug("Scheduled checkpointing for
partition {partition}. Offset={offset}", partition,
lastProcessEventArgs.Data.Offset);
                        await lastProcessEventArgs.UpdateCheckpointAsync();
                    }
                }
            }
        }
        catch (Exception e)
        {
            _logger.LogError(e, "Exception during checkpointing loop for
partition={lastPartition}", lastPartition);
        }
    }
}

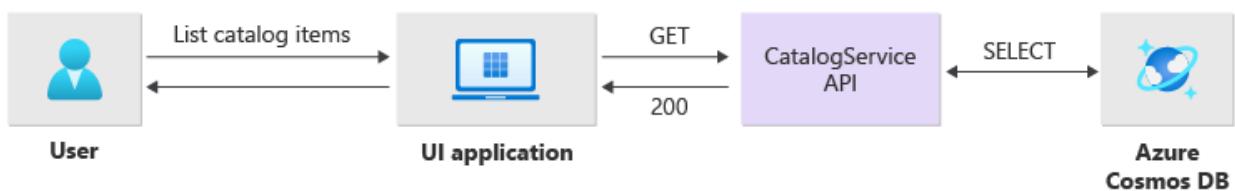
```

If the processor application encounters an error or is stopped before processing the message, then:

- Another instance will pick up the message for reprocessing, because it wasn't properly checkpointed in Storage.
- If the previous worker managed to persist the document in the database before failing, a conflict will happen (because the same ID and partition key is used) and the processor can safely ignore the message, as it has been already persisted.
- If the previous worker was terminated before writing to the database, new instance will repeat the steps and finalize persistence.

Implementation details for read operations

Read operations are processed directly by the API and immediately return data back to the user.



There is no back channel that communicates to the client if the operation completed successfully. The client application has to proactively poll the API to for updates of the item specified in the `Location` HTTP header.

Scalability

Individual workload components should scale out independently because each has different load patterns. The scaling requirements depend on the functionality of the service. Some services have a direct impact on end user and are expected to be able to scale out aggressively to provide fast response for a positive user experience and performance at any time.

In the implementation, the services are packaged as Docker containers and deployed by using Helm charts to each stamp. They are configured to have the expected Kubernetes requests and limits and a pre-configured auto-scaling rule in place. The `CatalogService` and the `BackgroundProcessor` workload component can scale in and out individually, both services are stateless.

End users interact directly with the `CatalogService`, so this part of the workload must respond under any load. There are at least 3 instances per cluster to spread across three Availability Zones in an Azure region. AKS horizontal pod autoscaler (HPA) takes care of automatically adding more pods if needed and Azure Cosmos DB auto-scale is able to

dynamically increase and reduce RUs available for the collection. Together, the `CatalogService` and Azure Cosmos DB form a **scale unit** within a stamp.

HPA is deployed with a Helm chart with configurable maximum and minimum number of replicas. The values are configured as:

During a load test it was identified that each instance is expected to handle ~250 requests/second with a standard usage pattern.

The `BackgroundProcessor` service has very different requirements and is considered a background worker which has limited impact on the user experience. As such, `BackgroundProcessor` has a different auto-scaling configuration than `CatalogService` and it can scale between 2 and 32 instances (this limit should be based on the number of partitions used in the Event Hubs - there's no benefit in having more workers than partitions).

Component	minReplicas	maxReplicas
CatalogService	3	20
BackgroundProcessor	2	32

In addition to that, each component of the workload including dependencies like `ingress-nginx` has [Pod Disruption Budgets \(PDBs\)](#) configured to ensure that a minimum number of instances is always available when changes are rolled out on clusters.

```
yml

#
# /src/app/charts/healthservice/templates/pdb.yaml
# Example pod distribution budget configuration.
#
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: {{ .Chart.Name }}-pdb
spec:
  minAvailable: 1
  selector:
    matchLabels:
      app: {{ .Chart.Name }}
```

ⓘ Note

The actual minimum and maximum number of pods for each component should be determined through load testing and can differ per workload.

Instrumentation

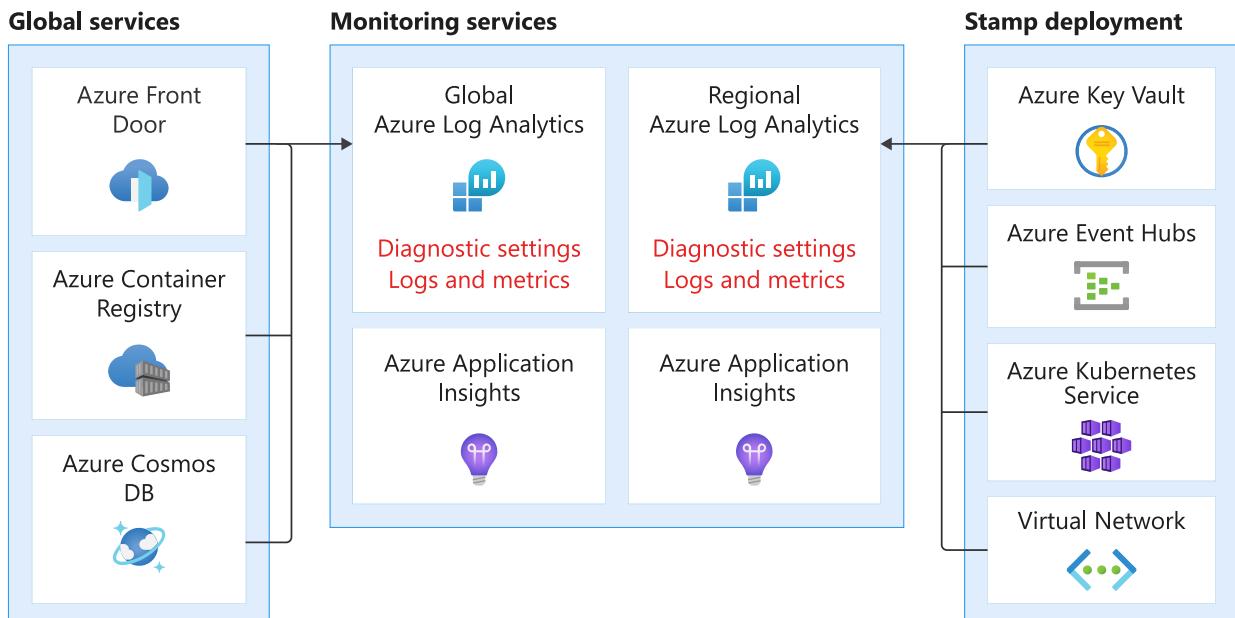
Instrumentation is an important mechanism in evaluating performance bottle necks and health issues that workload components can introduce in the system. Each component should emit sufficient information through metrics and trace logs to help quantify decisions. Here are some key considerations for instrumenting your application.

- Send logs, metrics and additional telemetry to the stamp's log system.
- Use structured logging instead of plain text so that information can be queried.
- Implement event correlation to ensure end-to-end transaction view. In the RI, every API response contains **Operation ID** as an HTTP header for traceability.
- Don't rely only on *stdout* (console) logging. However, these logs can be used for immediate troubleshooting of a failing pod.

This architecture implements distributed tracing with Application Insights backed by Log Analytics Workspace for all application monitoring data. Azure Log Analytics is used for logs and metrics of all workload and infrastructure components. The workload implements **full end-to-end tracing** of requests coming from the API, through Event Hubs, to Azure Cosmos DB.

ⓘ Important

Stamp monitoring resources are deployed to a separate monitoring resource group and have different lifecycle than the stamp itself. For more information, see [Monitoring data for stamp resources](#).



Implementation details for application monitoring

The `BackgroundProcessor` component uses the `Microsoft.ApplicationInsights.WorkerService` NuGet package to get out-of-the-box instrumentation from the application. Also, Serilog is used for all logging inside the application with Azure Application Insights configured as a sink (next to the console sink). Only when needed to track additional metrics, a `TelemetryClient` instance for Application Insights is used directly.

C#

```
//  
// /src/app/AlwaysOn.BackgroundProcessor/Program.cs  
  
public static IHostBuilder CreateHostBuilder(string[] args) =>  
    Host.CreateDefaultBuilder(args)  
        .ConfigureServices((hostContext, services) =>  
    {  
        Log.Logger = new LoggerConfiguration()  
  
            .ReadFrom.Configuration(hostContext.Configuration)  
                .Enrich.FromLogContext()  
                .WriteTo.Console(outputTemplate: "  
[{Timestamp:yyyy-MM-dd HH:mm:ss.fff zzz} {Level:u3}] {Message:lj}  
{Properties:j}{NewLine}{Exception}")  
  
            .WriteTo.ApplicationInsights(hostContext.Configuration[SysConfiguration.ApplicationInsightsConnStringKeyName], TelemetryConverter.Traces)  
                .CreateLogger();  
    }  
}
```

afint2b2b-cluster	POST Comments/AddNewItemComment [itemId/version]	202	19.4 ms
afint2b2b-brazilsou-evhns	EventHubProducerClient.Send		18.9 ms
afint2b2b-brazilsou-evhns	EventHubs.Message		13.1 µs
Link to EventHubProducerClient.Send			18.9 ms
afint2b2b-brazilsou-evhns	EventHubProducerClient.Send		18.9 ms
Link to EventProcessor.Process			41 ms
Time Spent in Queue			41 ms
BackgroundProcessor-brazilsouth	EventProcessor.Process	200	6.0 ms
afint-global-cosmos-brazilsouth	Add Comment		5.7 ms

To demonstrate practical request traceability, every API request (successful or not) returns the Correlation ID header to the caller. With this identifier the **application**

support team is able to search Application Insights and get a detailed view of the full transaction.

```
C#  
  
//  
// /src/app/AlwaysOn.CatalogService/Startup.cs  
//  
app.Use(async (context, next) =>  
{  
    context.Response.OnStarting(o =>  
    {  
        if (o is HttpContext ctx)  
        {  
            // ... code omitted for brevity  
            context.Response.Headers.Add("X-Server-Location",  
sysConfig.AzureRegion);  
            context.Response.Headers.Add("X-Correlation-ID",  
Activity.Current?.RootId);  
            context.Response.Headers.Add("X-Requested-Api-Version",  
ctx.GetRequestedApiVersion()?.ToString());  
        }  
        return Task.CompletedTask;  
    }, context);  
    await next();  
});
```

ⓘ Note

The Application Insights SDK has adaptive sampling enabled by default. That means that not every request is sent to the cloud and searchable by ID. Mission-critical application teams need to be able to reliably trace every request, therefore **the reference implementation has adaptive sampling disabled in production environment.**

Kubernetes monitoring implementation details

Besides the use of diagnostic settings to send AKS logs and metrics to Log Analytics, AKS is also configured to use **Container Insights**. Enabling Container Insights deploys the OMSAgentForLinux via a Kubernetes DaemonSet on each of the nodes in AKS clusters. The OMSAgentForLinux is capable of collecting additional logs and metrics from within the Kubernetes cluster and sends them to its corresponding Log Analytics workspace. This contains more granular data about pods, deployments, services and the overall cluster health.

Extensive logging can negatively affect cost while providing no benefit. For this reason, **stdout log collection and Prometheus scraping is disabled** for the workload pods in the Container Insights configuration, because all traces are already captured through Application Insights - generating duplicate records.

YAML

```
#  
# /src/config/monitoring/container-azm-ms-agentconfig.yaml  
# This is just a snippet showing the relevant part.  
#  
[log_collection_settings]  
  [log_collection_settings.stdout]  
    enabled = false  
  
  exclude_namespaces = ["kube-system"]
```

See the [full configuration file](#) for reference.

Health monitoring

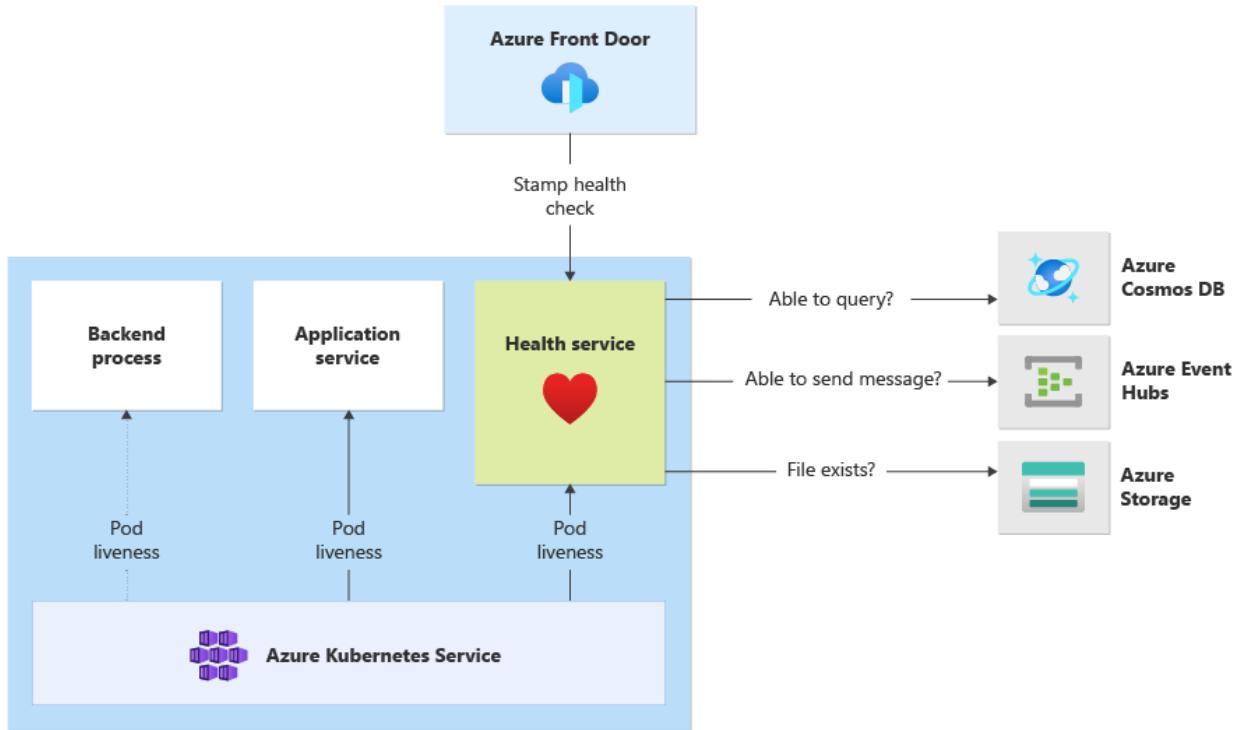
Application monitoring and observability are commonly used to quickly identify issues with a system and inform the [health model](#) about the current application state. Health monitoring, surfaced through *health endpoints* and used by *health probes* provides information, which is immediately actionable - typically instructing the main load balancer to take the unhealthy component out of rotation.

In the architecture, health monitoring is applied at these levels:

- Workload pods running on AKS. These pods have health and liveness probes, therefore AKS is able to manage their lifecycle.
- **Health service** is a dedicated component on the cluster. Azure Front Door is configured to probe health services in each stamp and remove unhealthy stamps from load balancing automatically.

Health service implementation details

`HealthService` is a workload component that is running along other components (`CatalogService` and `BackgroundProcessor`) on the compute cluster. It provides a REST API that is called by Azure Front Door health check to determine the availability of a stamp. Unlike basic liveness probes, health service is a more complex component which adds the state of dependencies in addition to its own.



If the AKS cluster is down, the health service won't respond, rendering the workload unhealthy. When the service is running, it performs periodic checks against critical components of the solution. All checks are done **asynchronously and in parallel**. If any of them fail, the whole stamp will be considered unavailable.

⚠ Warning

Azure Front Door health probes can generate significant load on the health service, because requests come from multiple PoP locations. To prevent overloading the downstream components, appropriate caching needs to take place.

The health service is also used for explicitly configured URL ping tests with each stamp's Application Insights resource.

For more details about the `HealthService` implementation, see [Application Health Service](#).

Networking and connectivity for mission-critical workloads

Article • 11/30/2023

The regional distribution of resources in the [mission-critical reference architecture](#) requires a robust network infrastructure.

A globally distributed design is recommended where Azure services come together to provide a highly available application. The global load balancer combined with regional stamps provides that guarantee through reliable connectivity.

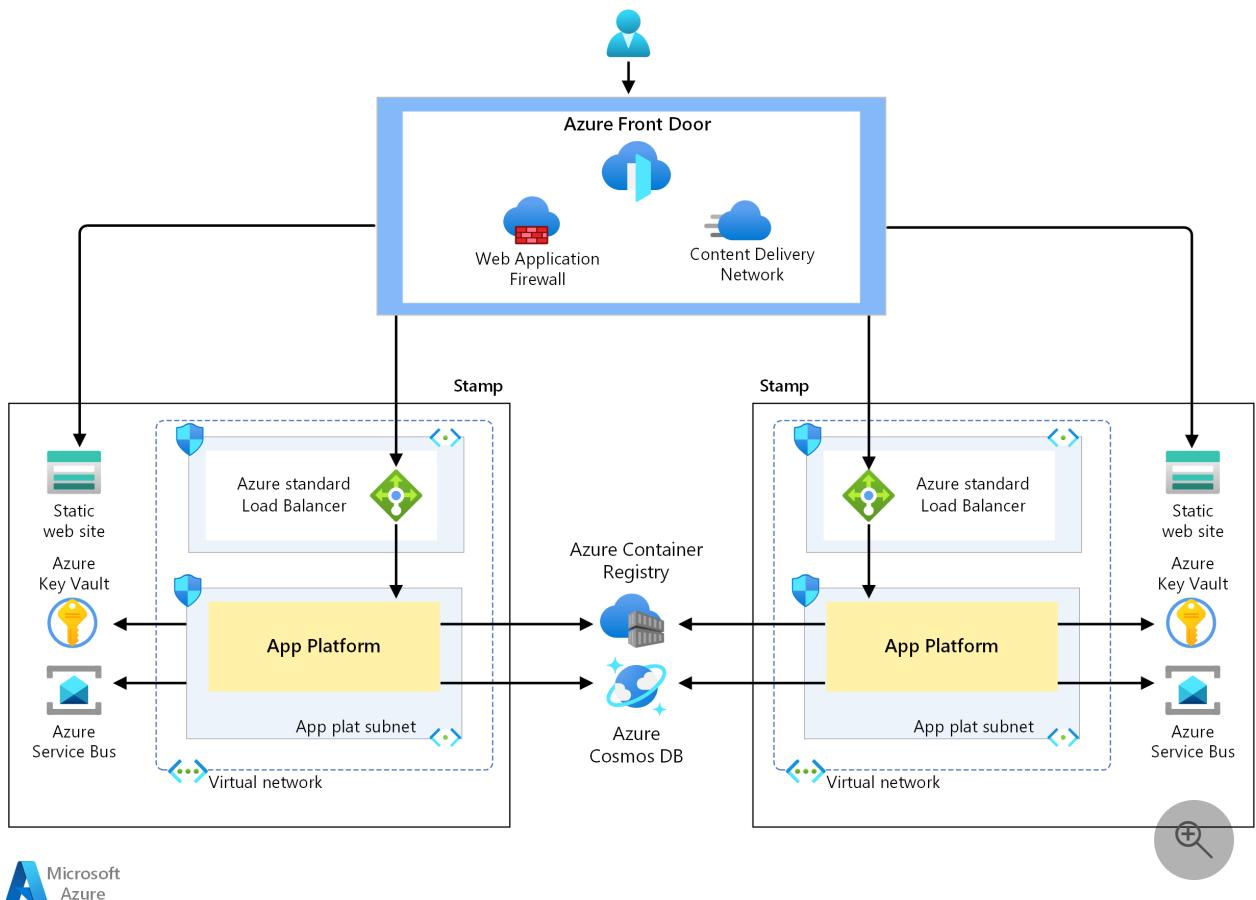
The regional stamps are the deployable unit in the architecture. The ability to quickly deploy a new stamp provides scalability and supports high availability. The stamps follow an isolated [virtual network design](#). Cross-stamp traffic isn't recommended. Virtual network peerings or VPN connections to other stamps aren't required.

The architecture is intentional in defining the regional stamps as short-lived. The global state of the infrastructure is stored in the global resources.

A global load balancer is required to route traffic to healthy stamps and provide security services. It must have certain capabilities.

- Health probing is required so that the load balancer can check the health of the origin before routing traffic.
- Weighted traffic distribution.

Optionally, it should be able to perform caching at the edge. Also, provide some security assurance for ingress through the use of web application firewall (WAF).



[Download a Visio file](#) of this architecture.

Traffic ingress

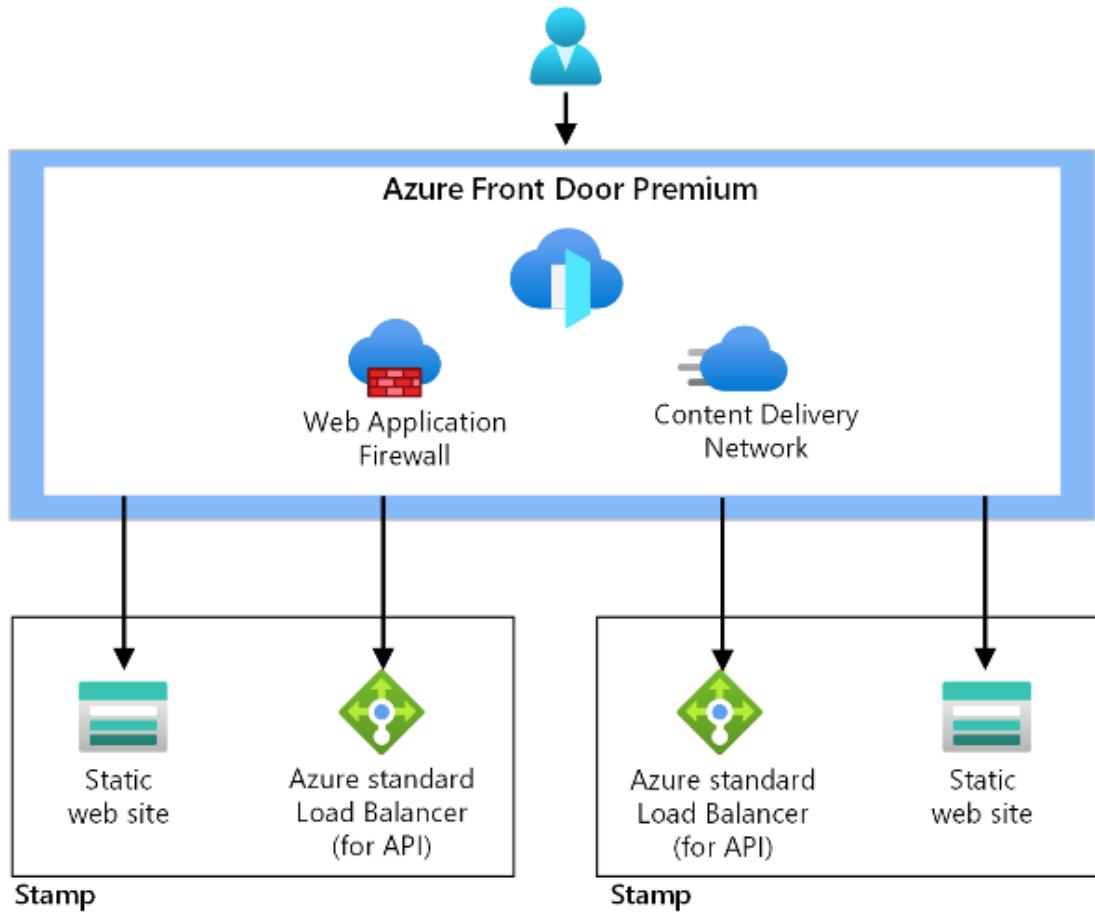
The application defined in the architecture is internet facing and has several requirements:

- A routing solution that is global and can distribute traffic between independent regional stamps.
- Low-latency in health checking and the ability to stop sending traffic to unhealthy stamps.
- Prevention of malicious attacks at the edge.
- Provide caching abilities at the edge.

The entry point for all traffic in the design is through Azure Front Door. Front Door is a global load balancer that routes HTTP(S) traffic to registered backends/origins. Front Door uses health probes that issue requests to a URI in each backend/origin. In the reference implementation, the URI called is a health service. The health service advertises the health of the stamp. Front Door uses the response to determine the health of an individual stamp and route traffic to healthy stamps capable of servicing application requests.

Azure Front Door integration with Azure Monitor provides near real-time monitoring of traffic, security, success and failure metrics, and alerting.

Azure Web Application Firewall, integrated with Azure Front Door, is used to prevent attacks at the edge before they enter the network.



Isolated virtual network - API

The API in the architecture uses Azure Virtual Networks as the traffic isolation boundary. Components in one virtual network can't communicate directly with components in another virtual network.

Requests to the application platform are distributed with a standard SKU external Azure Load Balancer. There is a check to ensure that traffic reaching the load balancer was routed via Azure Front Door. This check ensures that all traffic was inspected by the Azure WAF.

Build agents used for the operations and deployment of the architecture must be able to reach into the isolated network. The isolated network can be opened up to allow the

agents to communicate. Alternatively, self-hosted agents can be deployed in the virtual network.

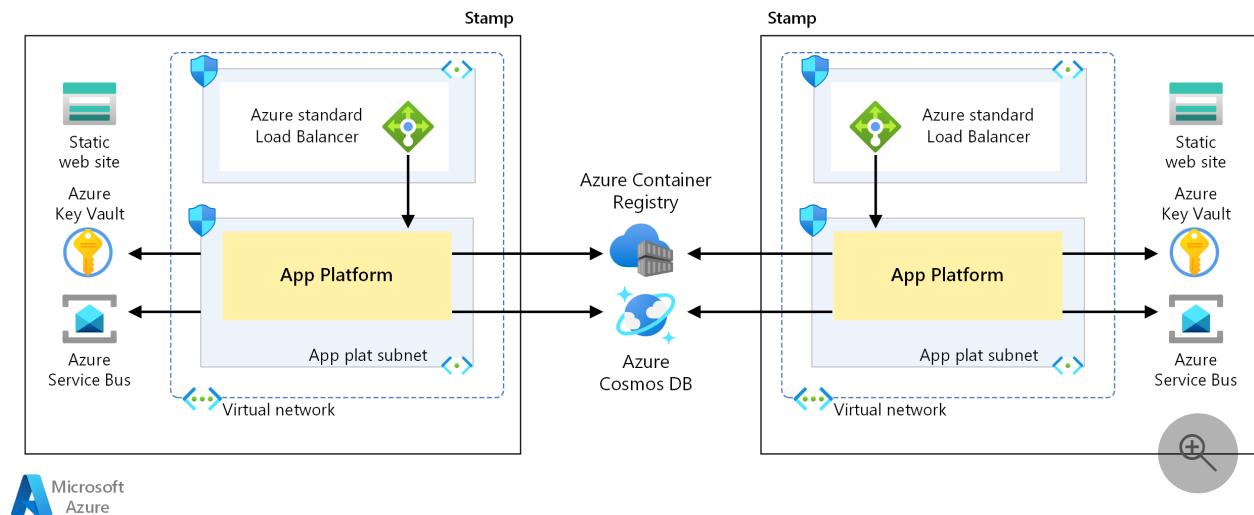
Monitoring of the network throughput, performance of the individual components, and health of the application is required.

Application platform communication dependency

The application platform used with the individual stamps in the infrastructure, has the following communication requirements:

- The application platform must be able to communicate securely with Microsoft PaaS services.
- The application platform must be able to communicate securely with other services when needed.

The architecture as defined uses Azure Key Vault to store secrets, such as connection strings and API keys, to securely communicate over the internet to Azure PaaS services. There are possible risks to exposing the application platform over the internet for this communication. Secrets can be compromised and increased security and monitoring of the public endpoints is recommended.



Extended networking considerations

This section discusses the pros and cons of alternative approaches to the network design. Alternative networking considerations and the use of Azure Private endpoints is the focus in the following sections.

Subnets and NSG

Subnets within the virtual networks can be used to segment traffic within the design. Subnet isolation separates resources for different functions.

Network security groups can be used control the traffic that is allowed in and out of each subnet. Rules used within the NSGs can be used limit traffic based on IP, port, and protocol to block unwanted traffic into the subnet.

Private endpoints - Ingress

The premium SKU of Front Door supports the use of Azure Private Endpoints. Private endpoints expose an Azure service to a private IP address in a virtual network.

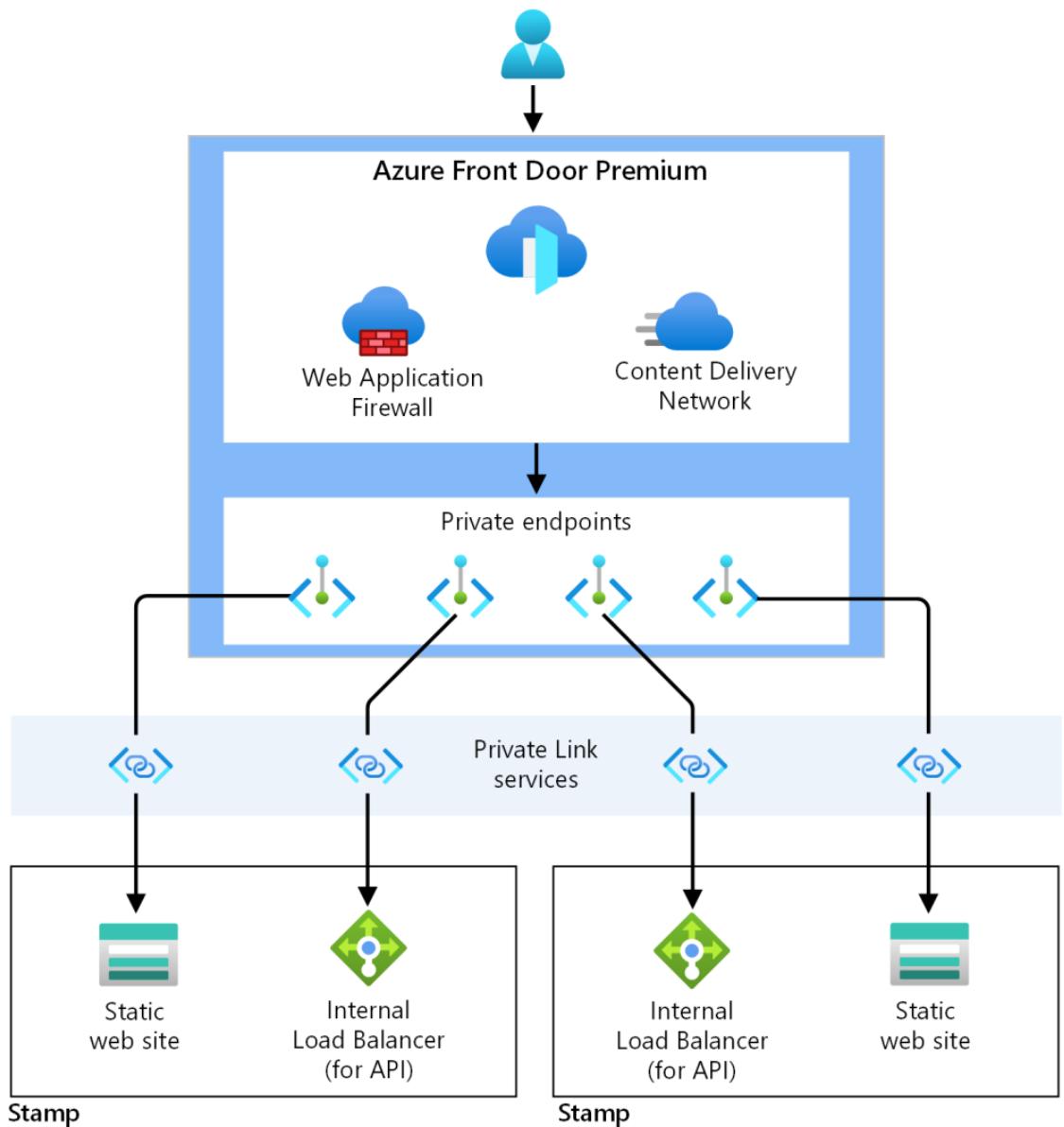
Connections are made securely and privately between services without the need to route the traffic to public endpoints.

Azure Front Door premium and Azure Private Endpoints enable fully private compute clusters in the individual stamps. Traffic is fully locked down for all Azure PaaS services.

Using private endpoints increases the security of the design. However, it introduces another point of failure. Public endpoints exposed in the application stamps are no longer needed and can no longer be accessed and exposed to a possible DDoS attack.

The increased security must be weighed versus the increased reliability effort, cost, and complexity.

Self-hosted build agents must be used for the stamp deployment. The management of these agents comes with a maintenance overhead.

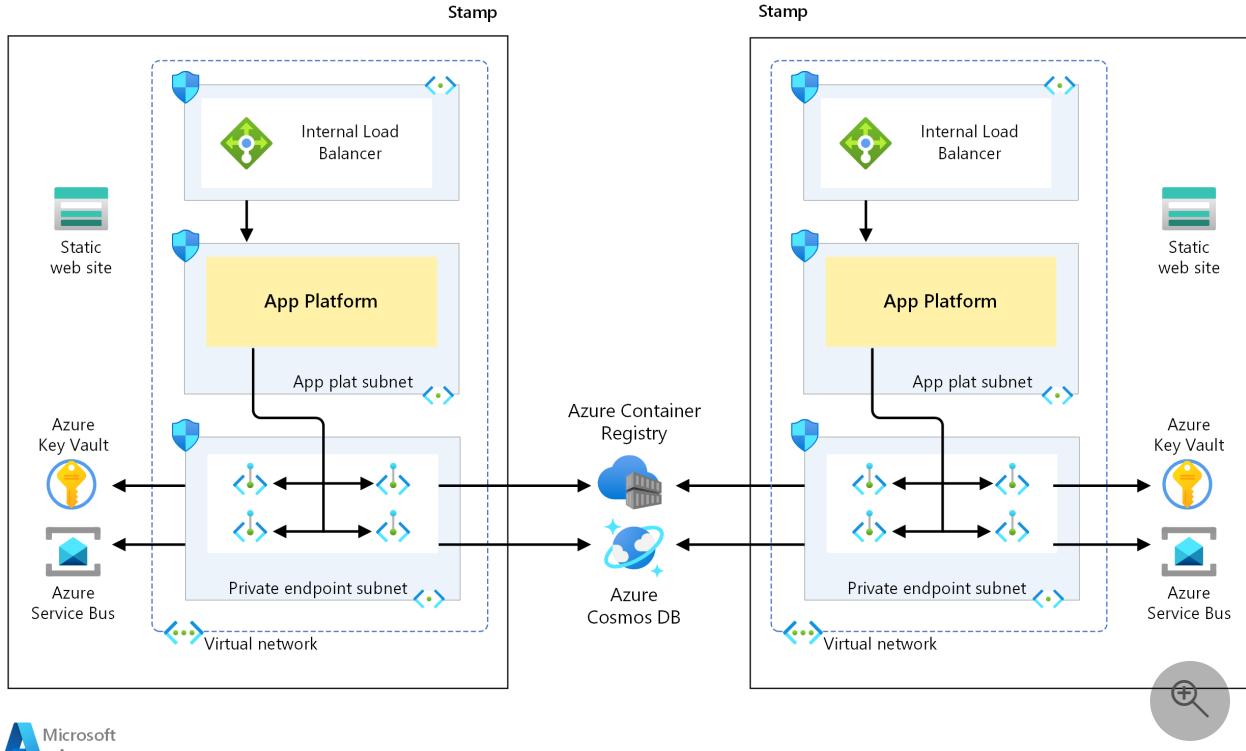


Private endpoints - Application platform

Private endpoints are supported for all Azure PaaS services used in this design. With private endpoints configured for the application platform, all communication would travel through the virtual network of the stamp.

The public endpoints of the individual Azure PaaS services can be configured to disallow public access. This would isolate the resources from public attacks that could cause downtime and throttling which affect reliability and availability.

Self-hosted build agents must be used for the stamp deployment the same as above. The management of these agents comes with a maintenance overhead.



Next steps

Deploy the reference implementation to get a full understanding of the resources and their configuration used in this architecture.

[Implementation: Mission-Critical Online](#)

Data platform for mission-critical workloads on Azure

Article • 12/01/2023

In a mission-critical architecture, any state must be stored outside the compute as much as possible. The choice of technology is based on these key architectural characteristics:

[+] Expand table

Characteristics	Considerations
Performance	How much compute is required?
Latency	What impact will the distance between the user and the data store have on latency? What is the desired level of consistency with tradeoff on latency?
Responsiveness	Is the data store required to be always available?
Scalability	What's the partitioning scheme?
Durability	Is the data expected to long lasting? What is the retention policy?
Resiliency	In case of a failure, is the data store able to fail over automatically? What measures are in place to reduce the failover time?
Security	Is the data encrypted? Can the datastore be reached over public network?

In this architecture, there are two data stores:

- **Database**

Stores related to the workload. It's recommended that all state is stored globally in a database separated from regional stamps. Build redundancy by deploying the database across regions. For mission-critical workloads, synchronizing data across regions should be the primary concern. Also, in case of a failure, write requests to the database should still be functional.

Data replication in an active-active configuration is highly recommended. The application should be able to instantly connect with another region. All instances should be able to handle read and write requests.

- **Message broker**

The only stateful service in the regional stamp is the message broker, which stores requests for a short period. The broker serves the need for buffering and reliable

messaging. The processed requests are persisted in the global database.

For other data considerations, see [Mission-critical guidance in Well-architected Framework: Data platform considerations](#).

Database

This architecture uses Azure Cosmos DB for NoSQL. This option is chosen because it provides the most features needed in this design:

- **Multi-region write**

Multi-region write is enabled with replicas deployed to every region in which a stamp is deployed. Each stamp can write locally and Azure Cosmos DB handles data replication and synchronization between the stamps. This capability significantly lowers latency for geographically distributed end-users of the application. The Azure Mission-Critical reference implementation uses multi-master technology to provide maximum resiliency and availability.

Zone redundancy is also enabled within each replicated region.

For details on multi-region writes, see [Configure multi-region writes in your applications that use Azure Cosmos DB](#).

- **Conflict management**

With the ability to perform writes across multiple regions comes the necessity to adopt a conflict management model as simultaneous writes can introduce record conflicts. Last Writer Wins is the default model and is used for the Mission Critical design. The last writer, as defined by the associated timestamps of the records wins the conflict. Azure Cosmos DB for NoSQL also allows for a custom property to be defined.

- **Query optimization**

A general query efficiency recommendation for read-heavy containers with a high number of partitions is to add an equality filter with the itemID identified. An in-depth review of query optimization recommendations can be found at [Troubleshoot query issues when using Azure Cosmos DB](#).

- **Backup feature**

It's recommended that you use the native backup feature of Azure Cosmos DB for data protection. [Azure Cosmos DB backup feature](#) supports online backups and

on-demand data restore.

(!) Note

Most workloads aren't purely OLTP. There is an increasing demand for real-time reporting, such as running reports against the operational system. This is also referred to as HTAP (Hybrid Transactional and Analytical Processing). Azure Cosmos DB supports this capability via [Azure Synapse Link for Azure Cosmos DB](#).

Data model for the workload

Data model should be designed such that features offered by traditional relational databases aren't required. For example, foreign keys, strict row/column schema, views, and others.

The workload has these **data access characteristics**:

- Read pattern:
 - Point reads - Fetching a single record. Item ID and partition key is directly used for maximum optimization (1 RU per request).
 - List reads - Getting catalog items to display in a list. `FeedIterator` with limit on number of results is used.
- Write pattern:
 - Small writes - Requests usually insert a single or a small number of records in a transaction.
- Designed to handle high traffic from end-users with the ability to scale to handle traffic demand in the order of millions of users.
- Small payload or dataset size - usually in order of KB.
- Low response time (in order of milliseconds).
- Low latency (in order of milliseconds).

Configuration

Azure Cosmos DB is configured as follows:

- **Consistency level** is set to the default *Session consistency* because it's the most widely used level for single region and globally distributed applications. Weaker consistency with higher throughput isn't needed because of the asynchronous nature of write processing and doesn't require low latency on database write.

(!) Note

The *Session* consistency level offers a reasonable tradeoff for latency, availability and consistency guarantees for this specific application. It's important to understand that *Strong* consistency level isn't available for multi-master write databases.

- **Partition key** is set to `/id` for all collections. This decision is based on the usage pattern, which is mostly "*writing new documents with GUID as the ID*" and "*reading wide range of documents by IDs*". Providing the application code maintains its ID uniqueness, new data is evenly distributed into partitions by Azure Cosmos DB, enabling infinite scale.
- **Indexing policy** is configured on collections to optimize queries. To optimize RU cost and performance, a custom indexing policy is used. This policy only indexes the properties that are used in query predicates. For example, the application doesn't use the comment text field as a filter in queries. It was excluded from the custom indexing policy.

Here's an example from the implementation that shows indexing policy settings using Terraform:

```
indexing_policy {  
    excluded_path {  
        path = "/description/?"  
    }  
  
    excluded_path {  
        path = "/comments/text/?"  
    }  
  
    included_path {  
        path = "/"  
    }  
}
```

For information about connection from the application to Azure Cosmos DB in this architecture, see [Application platform considerations for mission-critical workloads](#)

Messaging services

Mission critical systems often utilize messaging services for message or event processing. These services promote loose coupling and act as a buffer that insulates the

system against unexpected spikes in demand.

- Azure Service Bus is recommended for message-based workloads when handling high-value messages.
- Azure Event Hubs is recommended for event-based systems that process high volumes of events or telemetry.

The following are design considerations and recommendations for Azure Service Bus premium and Azure Event Hubs in a mission critical architecture.

Handle load

The messaging system must be able to handle the required throughput (as in MB per second). Consider the following:

- The non-functional requirements (NFRs) of the system should specify the average message size and the peak number of messages/second each stamp must support. This information can be used to calculate the required peak MB/second per stamp.
- The impact of a failover must be considered when calculating the required peak MB/second per stamp.
- For Azure Service Bus, the NFRs should specify any advanced Service Bus features such as sessions and de-duping messages. These features will affect the throughput of Service Bus.
- The throughput of Service Bus with the required features should be calculated through testing as MB/second per Messaging Unit (MU). For more information about this topic, see [Service Bus premium and standard messaging tiers](#).
- The throughput of Azure Event Hubs should be calculated through testing as MB/second per throughput unit (TU) for the standard tier or processing unit (PU) for premium tier. For more information about this topic, see [Scaling with Event Hubs](#).
- The above calculations can be used to validate that the messaging service can handle the required load per stamp, and the required number of scale units required to meet that load.
- The operations section will discuss auto-scaling.

Every message must be processed

Azure Service Bus premium tier is the recommended solution for high-value messages for which processing must be guaranteed. The following are details regarding this requirement when using Azure Service Bus premium:

- To ensure that messages are properly transferred to and accepted by the broker, message producers should use one of the supported Service Bus API clients. Supported APIs will only return successfully from a send operation if the message was persisted on the queue/topic.
- To ensure messages on the bus are processed, you should use [PeekLock receive mode](#). This mode enables at-least once processing. The following outlines the process:
 - The message consumer receives the message to process.
 - The consumer is given an exclusive lock on the message for a given time duration.
 - If the consumer successfully processes the message, it sends an acknowledgment back to the broker, and the message is removed from the queue.
 - If an acknowledgment isn't received by the broker in the allotted time period, or the handler explicitly abandons the message, the exclusive lock is released. The message is then available for other consumers to process the message.
 - If a message is not successfully processed a configurable number of times, or the handler forwards the message to the [dead-letter queue](#).
 - To ensure that messages sent to the dead-letter queue are acted upon, the dead-letter queue should be monitored, and alerts should be set.
 - The system should have tooling for operators to be able to [inspect, correct, and resubmit messages](#).
- Because messages can potentially be processed more than one time, message handlers should be made idempotent.

Idempotent message processing

In [RFC 7231 ↗](#), the Hypertext Transfer Protocol states, "A ... method is considered *idempotent* if the intended effect on the server of multiple identical requests with that method is the same as the effect for a single such request."

One common technique of making message handling idempotent is to check a persistent store, like a database, if the message has already been processed. If it has been processed, you wouldn't run the logic to process it again.

- There might be situations where the processing of the message includes database operations, specifically the insertion of new records with database-generated identifiers. New messages can be emitted to the broker, which contain those identifiers. Because there aren't distributed transactions that encompass both the database and the message broker, there can be a number of complications that

can occur if the process running the code happens to fail. See the following example situations:

- The code emitting the messages might run before the database transaction is committed, which is how many developers work using the [Unit of Work pattern](#). Those messages can *escape*, if the failure occurs between calling the broker and asking that the database transaction be committed. As the transaction rolls back, those database-generated IDs are also undone, which leaves them available to other code that might be running at the same time. This can cause recipients of the *escaped* messages to process the wrong database entries, which hurts the overall consistency and correctness of your system.
- If developers put the code that emits the message *after* the database transaction completes, the process can still fail between these operations (transaction committed - message sent). When that happens, the message will go through processing again, but this time the idempotence guard clause will see that it has already been processed (based on the data stored in the database). The clause will skip the message emitting code, believing that everything was done successfully last time. Downstream systems, which were expecting to receive notifications about the completed process, do not receive anything. This situation again results in an overall state of inconsistency.
- The solution to the above problems involves using the [Transactional Outbox pattern](#), where the outgoing messages are stored *off to the side*, in the same transactional store as the business data. The messages are then transmitted to the message broker, when the initial message has been successfully processed.
- Since many developers are unfamiliar with these problems or their solutions, in order to guarantee that these techniques are applied consistently in a mission-critical system, we suggest you have the functionality of the outbox and the interaction with the message broker wrapped in some kind of library. All code processing and sending transactionally-significant messages should make use of that library, rather than interacting with the message broker directly.
 - Libraries that implement this functionality in .NET include [NServiceBus](#) and [MassTransit](#).

High availability and disaster recovery

The message broker must be available for producers to send messages and consumers to receive them. The following are details regarding this requirement:

- To ensure the highest availability with Service Bus, use the premium tier, which has support for availability zones in supporting regions. With availability zones,

messages and metadata are replicated across three disparate data centers in the same region.

- Use supported Service Bus or Event Hubs SDKs to automatically retry read or write failures.
- Consider [active-active replication](#) or [active-passive replication](#) patterns to insulate against regional disasters.

 **Note**

Azure Service Bus Geo-disaster recovery only replicates metadata across regions. This feature does not replicate messages.

Monitoring

The messaging system acts as a buffer between message producers and consumers.

There are key indicator types that you should monitor in a mission-critical system that provide valuable insights described below:

- **Throttling** - Throttling indicates that the system doesn't have the required resources to process the request. Both Service Bus and Event Hubs support monitoring throttled requests. You should alert on this indicator.
- **Queue depth** - A queue depth that is growing can indicate that message processors aren't working or there aren't enough processors to handle the current load. Queue depth can be used to inform auto-scaling logic of handlers.
 - For Service Bus, queue depth is exposed as message count
 - For Event Hubs, the consumers have to calculate queue depth per partition and push the metric to your monitoring software. For each read, the consumer gets the sequence number of the current event, and the event properties of the last enqueued event. The consumer can calculate the offset.
- **Dead-letter queue** - Messages in the dead-letter queue represent messages that couldn't be processed. These messages usually require manual intervention. Alerts should be set on the dead-letter queue.
 - Service Bus has a [dead-letter queue](#) and a DeadLetterMessages metric.
 - For Event Hubs, this functionality must be custom logic built into the consumer.
- **CPU/Memory usage** - CPU and memory should be monitored to ensure the messaging system has enough resources to process the current load. Both Service Bus premium and Event Hubs premium expose CPU and memory Usage.
 - Messaging units (MUs) are used in Service Bus to isolate resources such as CPU and memory for a namespace. CPU and memory rising above a threshold can indicate that there aren't enough MUs configured, while falling below other

thresholds can indicate that there are too many MUs configured. These indicators can be used to [auto-scale MUs](#).

- Event Hubs premium tier uses processing units (PUs) to isolate resources, while standard tier uses throughput units (TUs). Those tiers don't require interaction with CPU/Memory to auto-inflate PUs or TUs.

Health check

The health of the messaging system must be considered in the health checks for a mission critical application. Consider the following factors:

- The messaging system acts as a buffer between message producers and consumers. The stamp can be viewed as healthy if producers are able to successfully send messages to the broker and if consumers are able to successfully process messages from the broker.
- The health check should ensure that messages can be sent to the message system.

Next steps

Deploy the reference implementation to get a full understanding of the resources and their configuration used in this architecture.

[Implementation: Mission-Critical Online](#)

Deployment and testing for mission-critical workloads on Azure

Article • 11/30/2023

The deployment and testing of the mission critical environment is a crucial piece of the overall reference architecture. The individual application stamps are deployed using infrastructure as code from a source code repository. Updates to the infrastructure, and the application on top, should be deployed with zero downtime to the application. A DevOps continuous integration pipeline is recommended to retrieve the source code from the repository and deploy the individual stamps in Azure.

Deployment and updates are the central process in the architecture. Infrastructure and application related updates should be deployed to fully independent stamps. Only the global infrastructure components in the architecture are shared across the stamps. Existing stamps in the infrastructure aren't touched. Infrastructure updates will only be deployed to these new stamps. Likewise, the new application version will only be deployed to these new stamps.

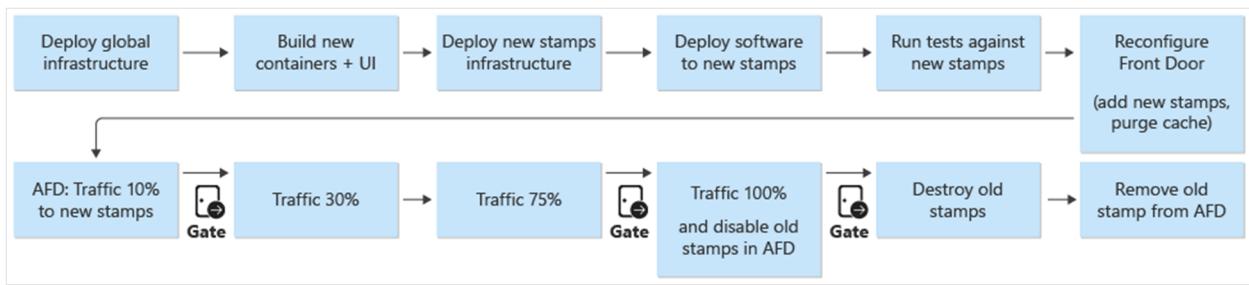
The new stamps are added to Azure Front Door. Traffic is gradually moved over to the new stamps. When it's determined that traffic is served from the new stamps without issue, the previous stamps are deleted.

Penetration, chaos, and stress testing are recommended for the deployed environment. Proactive testing of the infrastructure discovers weaknesses and how the deployed application will behave if there's a failure.

Deployment

The deployment of the infrastructure in the reference architecture is dependent upon the following processes and components:

- **DevOps** - The source code from GitHub and pipelines for the infrastructure.
- **Zero downtime updates** - Updates and upgrades are deployed to the environment with zero downtime to the deployed application.
- **Environments** - Short-lived and permanent environments used for the architecture.
- **Shared and dedicated resources** - Azure resources that are dedicated and shared to the stamps and overall infrastructure.



For more information, see [Deployment and testing for mission-critical workloads on Azure: Design considerations](#)

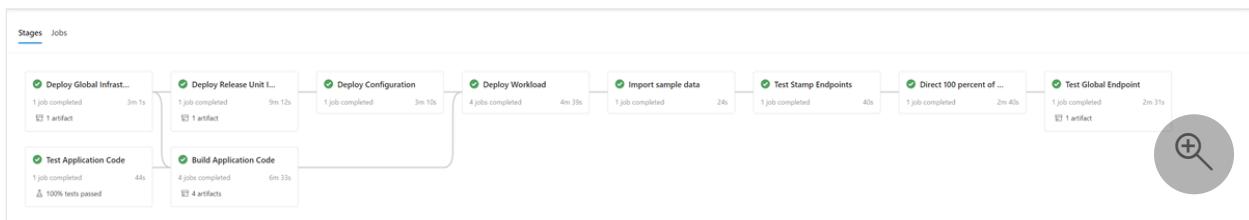
Deployment: DevOps

The DevOps components provide the source code repository and CI/CD pipelines for deployment of the infrastructure and updates. GitHub and Azure Pipelines were chosen as the components.

- **GitHub** - Contains the source code repositories for the application and infrastructure.
- **Azure Pipelines** - The pipelines used by the architecture for all build, test and release tasks.

An extra component in the design used for the deployment is build agents. Microsoft Hosted build agents are used as part of Azure Pipelines to deploy the infrastructure and updates. The use of Microsoft Hosted build agents removes the management burden for developers to maintain and update the build agent.

For more information about Azure Pipelines, see [What is Azure DevOps Services?](#).



For more information, see [Deployment and testing for mission-critical workloads on Azure: Infrastructure-as-Code deployments](#)

Deployment: Zero downtime updates

The zero-downtime update strategy in the reference architecture is central to the overall mission critical application. The methodology of replace instead of upgrade of the stamps ensures a fresh installation of the application into an infrastructure stamp. The

reference architecture utilizes a blue/green approach and allows for separate test and development environments.

There are two main components of the reference architecture:

- **Infrastructure** - Azure services and resources. Deployed with Terraform and its associated configuration.
- **Application** - The hosted service or application that serves users. Based on Docker containers and npm built artifacts in HTML and JavaScript for the single-page application (SPA) UI.

In many systems, there's an assumption that application updates will be more frequent than infrastructure updates. As a result, different update procedures are developed for each. With a public cloud infrastructure, changes can happen at a faster pace. One deployment process for application updates and infrastructure updates was chosen. One approach ensures infrastructure and application updates are always in sync. This approach allows for:

- **One consistent process** - Fewer chances for mistakes if infrastructure and application updates are mixed together in a release, intentionally or not.
- **Enables Blue/Green deployment** - Every update is deployed using a gradual migration of traffic to the new release.
- **Easier deployment and debugging of the application** - The entire stamp will never host multiple versions of the application side-by-side.
- **Simple rollback** - Traffic can be switched back to the stamps that run the previous version if errors or issues are encountered.
- **Elimination of manual changes and configuration drift** - Every environment is a fresh deployment.

For more information, see [Deployment and testing for mission-critical workloads on Azure: Ephemeral blue/green deployments](#)

Branching strategy

The foundation of the update strategy is the use of branches within the Git repository. The reference architecture uses three types of branches:

[+] Expand table

Branch	Description
<code>feature/*</code> and <code>fix/*</code>	The entry points for any change. These branches are created by developers and should be given a descriptive name, like <code>feature/catalog-update</code> or <code>fix/worker-timeout-bug</code> . When changes are ready to be merged, a pull request (PR) against the <code>main</code> branch is created. Every PR must be approved by at least one reviewer. With limited exceptions, every change that is proposed in a PR must run through the end-to-end (E2E) validation pipeline. The E2E pipeline should be used by developers to test and debug changes to a complete environment.
<code>main</code>	The continuously forward moving and stable branch. Mostly used for integration testing. Changes to <code>main</code> are made only through pull requests. A branch policy prohibits direct writes. Nightly releases against the permanent <code>integration (int)</code> environment are automatically executed from the <code>main</code> branch. The <code>main</code> branch is considered stable. It should be safe to assume that at any given time, a release can be created from it.
<code>release/*</code>	Release branches are only created from the <code>main</code> branch. The branches follow the format <code>release/2021.7.x</code> . Branch policies are used so that only repo administrators are allowed to create <code>release/*</code> branches. Only these branches are used to deploy to the <code>prod</code> environment.

For more information, see [Deployment and testing for mission-critical workloads on Azure: Branching strategy](#)

Hotfixes

When a hotfix is required urgently because of a bug or other issue and can't go through the regular release process, a hotfix path is available. Critical security updates and fixes to the user experience that weren't discovered during initial testing are considered valid examples of hotfixes.

The hotfix must be created in a new `fix` branch and then merged into `main` using a regular PR. Instead of creating a new release branch, the hotfix is "cherry-picked" into an existing release branch. This branch is already deployed to the `prod` environment. The CI/CD pipeline that originally deployed the release branch with all the tests is executed again and will now deploy the hotfix as part of the pipeline.

To avoid major issues, it's important that the hotfix contains a few isolated commits that can easily be cherry-picked and integrated into the release branch. If isolated commits can't be cherry-picked to integrate into the release branch, it's an indication that the change doesn't qualify as a hotfix. The change should be deployed as a full new release and potentially combined with a rollback to a former stable version until the new release can be deployed.

Deployment: Environments

The reference architecture uses two types of environments for the infrastructure:

- **Short-lived** - The E2E validation pipeline is used to deploy short-lived environments. Short-lived environments are used for pure validation or debugging environments for developers. Validation environments can be created from the `feature/*` branch, subjected to tests, and then destroyed if all tests were successful. Debugging environments are deployed in the same way as validation, but aren't destroyed immediately. These environments shouldn't exist for more than a few days and should be deleted when the corresponding PR of the feature branch is merged.
- **Permanent** - In the permanent environments there are `integration (int)` and `production (prod)` versions. These environments live continuously and aren't destroyed. The environments use fixed domain names like `int.mission-critical.app`. In a real world implementation of the reference architecture, a `staging` (pre-prod) environment should be added. The `staging` environment is used to deploy and validate `release` branches with the same update process as `prod` (Blue/Green deployment).
 - **Integration (int)** - The `int` version is deployed nightly from the `main` branch with the same process as `prod`. The switchover of traffic is faster than the previous release unit. Instead of gradually switching traffic over multiple days, as in `prod`, the process for `int` completes within a few minutes or hours. This faster switchover ensures the updated environment is ready by the next morning. Old stamps are automatically deleted if all tests in the pipeline are successful.
 - **Production (prod)** - The `prod` version is only deployed from `release/*` branches. The traffic switchover uses more granular steps. A manual approval gate is between each step. Each release creates new regional stamps and deploys the new application version to the stamps. Existing stamps aren't touched in the process. The most important consideration for `prod` is that it should be "always on". No planned or unplanned downtime should ever occur. The only exception is foundational changes to the database layer. A planned maintenance window may be needed.

Deployment: Shared and dedicated resources

The permanent environments (`int` and `prod`) within the reference architecture have different types of resources depending on if they're shared with the entire infrastructure or dedicated to an individual stamp. Resources can be dedicated to a particular release and exist only until the next release unit has taken over.

Release units

A release unit is several regional stamps per specific release version. Stamps contain all the resources that aren't shared with the other stamps. These resources are virtual networks, Azure Kubernetes Service cluster, Event Hubs, and Azure Key Vault. Azure Cosmos DB and ACR are configured with Terraform data sources.

Globally shared resources

All resources shared between release units are defined in an independent Terraform template. These resources are Front Door, Azure Cosmos DB, Container registry (ACR), and the Log Analytics workspaces and other monitoring-related resources. These resources are deployed before the first regional stamp of a release unit is deployed. The resources are referenced in the Terraform templates for the stamps.

Front Door

While Front Door is a globally shared resource across stamps, its configuration is slightly different than the other global resources. Front Door must be reconfigured after a new stamp is deployed. Front Door must be reconfigured to gradually switch over traffic to the new stamps.

The backend configuration of Front Door can't be directly defined in the Terraform template. The configuration is inserted with Terraform variables. The variable values are constructed before the Terraform deployment is started.

The individual component configuration for the Front Door deployment is defined as:

- **Frontend** - Session affinity is configured to ensure users don't switch between different UI versions during a single session.
- **Origins** - Front Door is configured with two types of origin groups:
 1. An origin group for static storage that serves the UI. The group contains the website storage accounts from all currently active release units. Different weights can be assigned to the origins from different release units to

gradually move traffic to a newer unit. Each origin from a release unit should have the same weight assigned.

2. An origin group for the API, which is hosted on AKS. If there are release units with different API versions, then an API origin group exists for each release unit. If all release units offer the same compatible API, all origins are added to the same group and assigned different weights.

- **Routing rules** - There are two types of routing rules:

1. A routing rule for the UI that is linked to the UI storage origin group.
2. A routing rule for each API currently supported by the origins. For example:
`/api/1.0/*` and `/api/2.0/*`.

If a release introduces a new version of the backend APIs, the changes will reflect in the UI that is deployed as part of the release. A specific release of the UI will always call a specific version of the API URL. Users served by a UI version will automatically use the respective backend API. Specific routing rules are needed for different instances of the API version. These rules are linked to the corresponding origin groups. If a new API wasn't introduced, all API related routing rules link to the single origin group. In this case, it doesn't matter if a user is served the UI from a different release than the API.

Deployment: Deployment process

A blue/green deployment is the goal of the deployment process. A new release from a `release/*` branch is deployed into the `prod` environment. User traffic is gradually shifted to the stamps for the new release.

As a first step in the deployment process of a new version, the infrastructure for the new release unit is deployed with Terraform. Execution of the infrastructure deployment pipeline deploys the new infrastructure from a selected release branch. In parallel to the infrastructure provisioning, the container images are built or imported and pushed to the globally shared container registry (ACR). When the previous processes are completed, the application is deployed to the stamps. From the implementation viewpoint, it's one pipeline with multiple dependent stages. The same pipeline can be re-executed for hotfix deployments.

After the new release unit is deployed and validated, it's added to Front Door to receive user traffic.

A switch/parameter that distinguishes between releases that do and don't introduce a new API version should be planned for. Based on if the release introduces a new API version, a new origin group with the API backends must be created. Alternatively, new API backends can be added to an existing origin group. New UI storage accounts are added to the corresponding existing origin group. Weights for new origins should be set according to the desired traffic split. A new routing rule as described above must be created that corresponds to the appropriate origin group.

As a part of the addition of the new release unit, the weights of the new origins should be set to the desired minimum user traffic. If no issues are detected, the amount of user traffic should be increased to the new origin group over a period of time. To adjust the weight parameters, the same deployment steps should be executed again with the desired values.

Release unit teardown

As part of the deployment pipeline for a release unit, there's a destroy stage that removes all stamps once a release unit is no longer needed. All traffic is moved to a new release version. This stage includes the removal of release unit references from Front Door. This removal is critical to enable the release of a new version at a later date. Front Door must point to a single release unit in order to be prepared for the next release in the future.

Checklists

As part of the release cadence, a pre and post release checklist should be used. The following example is of items that should be in any checklist at a minimum.

- **Pre-release checklist** - Before starting a release, check the following:
 - Ensure the latest state of the `main` branch was successfully deployed to and tested in the `int` environment.
 - Update the changelog file through a PR against the `main` branch.
 - Create a `release/` branch from the `main` branch.
- **Post-release checklist** - Before old stamps are destroyed and their references are removed from Front Door, check that:
 - Clusters are no longer receiving incoming traffic.

- Event Hubs and other message queues don't contain any unprocessed messages.

Deployment: Limitations and risks of the update strategy

The update strategy described in this reference architecture has some limitations and risks that should be mentioned:

- Higher cost - When releasing updates, many of the infrastructure components are active twice for the release period.
- Front Door complexity - The update process in Front Door is complex to implement and maintain. The ability to execute effective blue/green deployments with zero downtime is dependent on it working properly.
- Small changes time consuming - The update process results in a longer release process for small changes. This limitation can be partially mitigated with the hotfix process described in the previous section.

Deployment: Application data forward compatibility considerations

The update strategy can support multiple versions of an API and work components executing concurrently. Because Azure Cosmos DB is shared between two or more versions, there's a possibility that data elements changed by one version might not always match the version of the API or workers consuming it. The API layers and workers must implement forward compatibility design. Earlier versions of the API or worker components processes data that was inserted by a later API or worker component version. It ignores parts it doesn't understand.

Testing

The reference architecture contains different tests used at different stages within the testing implementation.

These tests include:

- **Unit tests** - These tests validate that the business logic of the application works as expected. The reference architecture contains a sample suite of unit tests executed

automatically before every container build by Azure Pipelines. If any test fails, the pipeline will stop. Build and deployment won't proceed.

- **Load tests** - These tests help to evaluate the capacity, scalability, and potential bottlenecks for a given workload or stack. The reference implementation contains a user load generator to create synthetic load patterns that can be used to simulate real traffic. The load generator can also be used independently of the reference implementation.
- **Smoke tests** - These tests identify if the infrastructure and workload are available and act as expected. Smoke tests are executed as part of every deployment.
- **UI tests** - These tests validate that the user interface was deployed and works as expected. The current implementation only captures screenshots of several pages after deployment without any actual testing.
- **Failure injection tests** - These tests can be automated or executed manually. Automated testing in the architecture integrates Azure Chaos Studio as part of the deployment pipelines.

For more information, see [Deployment and testing for mission-critical workloads on Azure: Continuous validation and testing](#)

Testing: Frameworks

The online reference implementation existing testing capabilities and frameworks whenever possible.

 Expand table

Framework	Test	Description
NUnit	Unit	This framework is used for unit testing the .NET Core portion of the implementation. Unit tests are executed automatically by Azure Pipelines before container builds.
JMeter with Azure Load Testing	Load	Azure Load Testing is a managed service used to execute Apache JMeter load test definitions.
Locust	Load	Locust is an open-source load testing framework written in Python.
Playwright	UI and Smoke	Playwright is an open source Node.js library to automate Chromium, Firefox and WebKit with a single API. The Playwright

Framework	Test	Description
		test definition can also be used independently of the reference implementation.
Azure Chaos Studio	Failure injection	The reference implementation uses Azure Chaos Studio as an optional step in the E2E validation pipeline to inject failures for resiliency validation.

Testing: Failure Injection testing and Chaos Engineering

Distributed applications should be resilient to service and component outages. Failure Injection testing (also known as Fault Injection or Chaos Engineering) is the practice of subjecting applications and services to real-world stresses and failures.

Resilience is a property of an entire system and injecting faults helps to find issues in the application. Addressing these issues helps to validate application resiliency to unreliable conditions, missing dependencies and other errors.

Manual and automatic tests can be executed against the infrastructure to find faults and issues in the implementation.

Automatic

The reference architecture integrates [Azure Chaos Studio](#) to deploy and run a set of Azure Chaos Studio experiments to inject various faults at the stamp level. Chaos experiments can be executed as an optional part of the E2E deployment pipeline. When the tests are executed, the optional load test is always executed in parallel. The load test is used to create load on the cluster to validate the effect of the injected faults.

Manual

Manual failure injection testing should be done in an E2E validation environment. This environment ensures full representative tests without risk of interference from other environments. Most of the failures generated with the tests can be observed directly in the Application Insights [Live metrics](#) view. The remaining failures are available in the Failures view and corresponding log tables. Other failures require deeper debugging such as the use of `kubectl` to observe the behavior inside of AKS.

Two examples of failure injection tests performed against the reference architecture are:

- **DNS-based failure injection** - A test case that can simulate multiple issues. DNS resolution failures due to either the failure of a DNS server or Azure DNS. DNS based testing can help simulate general connections issues between a client and a service, for example when the **BackgroundProcessor** can't connect to the Event Hubs.

In single-host scenarios, you can modify the local `hosts` file to overwrite DNS resolution. In a larger environment with multiple dynamic servers like AKS, a `hosts` file isn't feasible. [Azure Private DNS Zones](#) can be used as an alternative to test failure scenarios.

Azure Event Hubs and Azure Cosmos DB are two of the Azure services used within the reference implementation that can be used to inject DNS-based failures. Event Hubs DNS resolution can be manipulated with an Azure Private DNS zone tied to the virtual network of one of the stamps. Azure Cosmos DB is a globally replicated service with specific regional endpoints. Manipulation of the DNS records for those endpoints can simulate a failure for a specific region and test the failover of clients.

- **Firewall blocking** - Most Azure services support firewall access restrictions based on virtual networks and/or IP addresses. In the reference infrastructure, these restrictions are used to restrict access to Azure Cosmos DB or Event Hubs. A simple procedure is to remove existing **Allow** rules or adding new **Block** rules. This procedure can simulate firewall misconfigurations or service outages.

The following example services in the reference implementation can be tested with a firewall test:

[] [Expand table](#)

Service	Result
Key Vault	When access to Key Vault is blocked, the most direct effect was the failure of new pods to spawn. The Key Vault CSI driver that fetches secrets on pod startup can't perform its tasks and prevents the pod from starting. Corresponding error messages can be observed with <code>kubectl describe po CatalogService-deploy-my-new-pod -n workload</code> . Existing pods will continue to work, although the same error message will be observed. The error message is generated by the results of the periodic update check for secrets. Though untested, it's assumed that executing a deployment won't work while Key Vault is inaccessible. Terraform and Azure CLI tasks within the pipeline run makes requests to Key Vault.
Event Hubs	When access to Event Hubs is blocked, new messages sent by the CatalogService and HealthService will fail. Retrieval of messages by the BackgroundProcess will slowly fail, with total failure within a few minutes.

Service	Result
Azure Cosmos DB	Removal of the existing firewall policy for a virtual network results in the Health Service to begin to fail with minimum lag. This procedure only simulates a specific case, an entire Azure Cosmos DB outage. Most failure cases that occur on a regional level should be mitigated automatically by transparent failover of the client to a different Azure Cosmos DB region. The DNS-based failure injection testing described previously is a more meaningful test for Azure Cosmos DB.
Container registry (ACR)	When the access to ACR is blocked, the creation of new pods that have been pulled and cached previously on an AKS node will continue to work. The creation still works due to the <code>k8s</code> deployment flag <code>pullPolicy=IfNotPresent</code> . Nodes that haven't pulled and cached an image before the block can't spawn a new pod and fails immediately with <code>ErrImagePull</code> errors. <code>kubectl describe pod</code> displays the corresponding <code>403 Forbidden</code> message.
AKS ingress Load Balancer	The alteration of the inbound rules for HTTP(S)(ports 80 and 443) in the AKS managed Network Security Group (NSG) to <code>Deny</code> results in user or health probe traffic fail to reach the cluster. The test of this failure is difficult to pinpoint the root cause, which was simulated as blockage between the network path of Front Door and a regional stamp. Front Door immediately detects this failure and takes the stamp out of rotation.

Health modeling for mission-critical workloads

Article • 11/30/2023

The monitoring of applications and infrastructure is an important part of any infrastructure deployment. For a mission-critical workload, monitoring is a critical part of the deployment. Monitoring application health and key metrics of Azure resources helps you understand if the environment is working as expected.

To fully understand these metrics and evaluate the overall health of a workload, requires a holistic understanding of all of the data monitored. A health model can assist with evaluation of the overall health status by displaying a clear indication of the health of the workload instead of raw metrics. The status is often presented as "traffic light" indicators such as red, green, or yellow. Representation of a health model with clear indicators makes it intuitive for an operator to understand the overall health of the workload and respond quickly to issues that arise.

Health modeling can be expanded into the following operational tasks for the mission-critical deployment:

- **Application Health Service** - Application component on the compute cluster that provides an API to determine the health of a stamp.
- **Monitoring** - Collection of performance and application counters that evaluate the health and performance of the application and infrastructure.
- **Alerting** - Actionable alerts of issues or outages in the infrastructure and application.
- **Failure analysis** - Breakdown and analysis of any failures including documentation of root cause.

These tasks make up a comprehensive health model for the mission-critical infrastructure. Development of a health model can and should be an exhaustive and integral part of any mission-critical deployment.

For more information, see [Health modeling and observability of mission-critical workloads on Azure](#).

Application Health Service

The Application Health Service (**HealthService**) is an application component that resides with the Catalog Service (**CatalogService**) and the Background Processor (**BackgroundProcessor**) within the compute cluster. The **HealthService** provides a REST API for Azure Front Door to call to determine the health of a stamp. The **HealthService** is a complex component that reflects the state of dependencies, in addition to its own.

When the compute cluster is down, the health service won't respond. When the services are up and running, it performs periodic checks against the following components in the infrastructure:

- It attempts to do a query against Azure Cosmos DB.
- It attempts to send a message to Event Hubs. The message is filtered out by the background worker.
- It looks up a state file in the storage account. This file can be used to turn off a region, even while the other checks are still operating correctly. This file can be used to communicate with other processes. For example, if the stamp is to be vacated for maintenance purposes, the file could be deleted in order to force an unhealthy state and reroute traffic.
- It queries the health model to determine if all operational metrics are within the predetermined thresholds. When the health model indicates the stamp is unhealthy, traffic shouldn't be routed to the stamp even though the other tests the **HealthService** performs return successfully. The Health Model takes a more complete view of the health status into account.

All health check results are cached in memory for a configurable number of seconds, by default 10. This operation does potentially add a small latency in detecting outages, but it ensures not every **HealthService** query requires backend calls, thus reducing load on the cluster and downstream services.

This caching pattern is important, because the number of **HealthService** queries grows significantly when using a global router like Azure Front Door: Every edge node in every Azure datacenter that serves requests will call the Health Service to determine if it has a functional backend connection. Caching the results reduces extra cluster load generated by health checks.

Configuration

The **HealthService** and the **CatalogService** have configuration settings common between the components except for the following settings used exclusively by the **HealthService**:

[\[\] Expand table](#)

Setting	Value
<code>HealthServiceCacheDurationSeconds</code>	Controls the expiration time of memory cache, in seconds.
<code>HealthServiceStorageConnectionString</code>	Connection string for the Storage Account where the status file should be present.
<code>HealthServiceBlobContainerName</code>	Storage Container where the status file should be present.
<code>HealthServiceBlobName</code>	Name of the status file - health check will look for this file.
<code>HealthServiceOverallTimeoutSeconds</code>	Timeout for the whole check - defaults to 3 seconds. If the check doesn't finish in this interval, the service reports unhealthy.

Implementation

All checks are performed asynchronously and **in parallel**. If either of them fails, **the whole stamp will be considered unavailable**.

Check results are cached in memory, using the standard, non-distributed ASP.NET Core `MemoryCache`. Cache expiration is controlled by `SysConfig.HealthServiceCacheDurationSeconds` and is set to 10 seconds by default.

ⓘ Note

The `SysConfig.HealthServiceCacheDurationSeconds` configuration setting reduces the additional load generated by health checks as not every request will result in downstream call to the dependent services.

The following table details the health checks for the components in the infrastructure:

[\[\] Expand table](#)

Component	Health check
Storage account	The blob check currently serves two purposes: <ol style="list-style-type: none">1. Test if it's possible to reach Blob Storage. The storage account is used by other components in the stamp and is considered a critical resource.2. Manually "turn off" a region by deleting the state file. A design decision was made that the check would only look for a presence of a state

Component	Health check
	<p>file in the specified blob container. The content of the file isn't processed. There's a possibility to set up a more sophisticated system that reads the content of the file and return different status based on the content of the file.</p> <p>Examples of content are HEALTHY, UNHEALTHY, and MAINTENANCE.</p> <p>Removal of the state file will disable the stamp. Ensure the health file is present after deploying the application. Absence of the health file will cause the service to always respond with UNHEALTHY. Front Door won't recognize the backend as available.</p> <p>The file is created by Terraform and should be present after the infrastructure deployment.</p>
Event Hub	<p>Event Hubs health reporting is handled by the <code>EventHubProducerService</code>. This service reports healthy if it's able to send a new message to Event Hubs. For filtering, this message has an identifying property added to it:</p> <p><code>HEALTHCHECK=TRUE</code></p> <p>This message is ignored on the receiving end. The <code>AlwaysOn.BackgroundProcessor.EventHubProcessorService.ProcessEventHandlerAsync()</code> configuration setting checks for the <code>HEALTHCHECK</code> property.</p>
Azure Cosmos DB	<p>Azure Cosmos DB health reporting is handled by the <code>CosmosDbService</code>, which reports healthy if it is:</p> <ol style="list-style-type: none"> 1. Able to connect to Azure Cosmos DB database and perform a query. 2. Able to write a test document to the database. <p>The test document has a short Time-to-Live set, Azure Cosmos DB automatically removes it.</p> <p>The HealthService performs two separate probes. If Azure Cosmos DB is in a state where reads work and writes don't, the two probes ensure an alert is triggered.</p>

Azure Cosmos DB queries

For the Read-only query, the following query is being used, which doesn't fetch any data and doesn't have a large effect on overall load:

SQL

```
SELECT GetCurrentDateTime ()
```

The write query creates a dummy `ItemRating` with minimum content:

C#

```
var testRating = new ItemRating()
{
    Id = Guid.NewGuid(),
    CatalogItemId = Guid.NewGuid(), // Create some random (=non-existing)
    item id
    CreationDate = DateTime.UtcNow,
```

```
    Rating = 1,
    TimeToLive = 10 // will be auto-deleted after 10sec
};

await AddNewRatingAsync(testRating);
```

Monitoring

Azure Log Analytics is used as the central store for logs and metrics for all application and infrastructure components. Azure Application Insights is used for all application monitoring data. Each stamp in the infrastructure has a dedicated Log Analytics workspace and Application Insights instance. A separate Log Analytics workspace is used for the globally shared resources such as Front Door and Azure Cosmos DB.

All stamps are short-lived and continuously replaced with each new release. The per-stamp Log Analytics workspaces are deployed as a global resource in a separate monitoring resource group as the stamp Log Analytics resources. These resources don't share the lifecycle of a stamp.

For more information, see [Unified data sink for correlated analysis](#).

Monitoring: Data sources

- **Diagnostic settings:** All Azure services used for Azure Mission-Critical are configured to send all their Diagnostic data including logs and metrics to the deployment specific (global or stamp) Log Analytics Workspace. This process happens automatically as part of the Terraform deployment. New options will be identified automatically and added as part of `terraform apply`.
- **Kubernetes monitoring:** Diagnostic settings are used to send AKS logs and metrics to Log Analytics. AKS is configured to use **Container Insights**. Container Insights deploys the **OMSAgentForLinux** via a Kubernetes DaemonSet on each node in the AKS clusters. The OMSAgentForLinux is capable of collecting extra logs and metrics from within the Kubernetes cluster and sends them to its corresponding Log Analytics workspace. These extra logs and metrics contain more granular data about pods, deployments, services and the overall cluster health. To gain more insights from the various components like ingress-nginx, cert-manager, and other components deployed to Kubernetes next to the mission-critical workload, it's possible to use [Prometheus scraping](#). Prometheus scraping configures the OMSAgentForLinux to scrape Prometheus metrics from various endpoints within the cluster.

- **Application Insights telemetry:** Application Insights is used to collect telemetry data from the application. The code has been instrumented to collect data on the performance of the application with the Application Insights SDK. Critical information, such as the resulting status code and duration of dependency calls and counters for unhandled exceptions is collected. This information is used in the Health Model and is available for alerting and troubleshooting.

Monitoring: Application Insights availability tests

To monitor the availability of the individual stamps and the overall solution from an outside point of view, [Application Insights Availability Tests](#) are set up in two places:

- **Regional availability tests:** These tests are set up in the regional Application Insights instances and are used to monitor the availability of the stamps. These tests target the clusters and the static storage accounts of the stamps directly. To call the ingress points of the clusters directly, requests need to carry the correct Front Door ID header, otherwise they're rejected by the ingress controller.
- **Global availability test:** These tests are set up in the global Application Insights instance and are used to monitor the availability of the overall solution by pinging Front Door. Two tests are used: One to test an API call against the **CatalogService** and one to test the home page of the website.

Monitoring: Queries

Azure Mission-Critical uses different Kusto Query Language (KQL) queries to implement custom queries as functions to retrieve data from Log Analytics. These queries are stored as individual files in our code repository, separated for global and stamp deployments. They're imported and applied automatically via Terraform as part of each infrastructure pipeline run.

This approach separates the query logic from the visualization layer. The Log Analytics queries are called directly from code, for example from the **HealthService API**. Another example is from a visualization tool such as Azure Dashboards, Monitor Workbooks, or Grafana.

Monitoring: Visualization

For visualizing the results of our Log Analytics health queries, we've used Grafana in our reference implementation. Grafana is used to show the results of Log Analytics queries and doesn't contain any logic itself. The Grafana stack isn't part of the solution's deployment lifecycle, but released separately.

For more information, see [Visualization](#).

Alerting

Alerts are an important part of the overall operations strategy. Proactive monitoring such as the use of dashboards should be used with alerts that raise immediate attention to issues.

These alerts form an extension of the health model, by alerting the operator to a change in health state, either to degraded/yellow state or to unhealthy/red state. By setting the alert to the root node of the Health Model, the operator is immediately aware of any business level affect to the state of the solution: After all, this root node will turn yellow or red if any of the underlying user flows or resources report yellow or red metrics. The operator can direct their attention to the Health Model visualization for troubleshooting.

For more information, see [Automated incident response](#).

Failure analysis

Composing the failure analysis is mostly a theoretical planning exercise. This theoretical exercise should be used as input for the automated failure injections that are part of the continuous validation process. By simulating the failure modes defined here, we can validate the resiliency of the solution against these failures to ensure they won't lead to outages.

The following table lists example failure cases of the various components of the Azure Mission-Critical reference implementation.

[] Expand table

Service	Risk	Impact/Mitigation/Comment	Outage
Microsoft Entra ID	Microsoft Entra ID becomes unavailable.	Currently no possible mitigation in place. A multi-region approach won't mitigate any outages as it's a global service. This service is a hard dependency. Microsoft Entra ID is used for control plane	Partial

Service	Risk	Impact/Mitigation/Comment	Outage
		operations like the creation of new AKS nodes, pulling container images from ACR or to access Key Vault on pod startup. It's expected that existing, running components should be able to keep running when Microsoft Entra ID experiences issues. It's likely that new pods or AKS nodes will be unable to spawn. In scale operations are required during this time, it could lead to a decreased user experience and potentially to outages.	
Azure DNS	Azure DNS becomes unavailable and DNS resolution fails.	If Azure DNS becomes unavailable, the DNS resolution for user requests and between different components of the application will likely fail. Currently no possible mitigation in place for this scenario. A multi-region approach won't mitigate any outages as it's a global service. Azure DNS is a hard dependency. External DNS services as backup wouldn't help, since all the PaaS components used rely on Azure DNS. Bypassing DNS by switching to IP isn't an option. Azure services don't have static, guaranteed IP addresses.	Full
Front Door	General Front Door outage.	If Front Door goes down entirely, there's no mitigation. This service is a hard dependency.	Yes
Front Door	Routing/frontend/backend configuration errors.	Can happen due to mismatch in configuration when deploying. Should be caught in testing stages. Frontend configuration with DNS is specific to each environment. Mitigation: Rolling back to previous configuration should fix most issues.. As changes take a couple of minutes	Full

Service	Risk	Impact/Mitigation/Comment	Outage
		in Front Door to deploy, it will cause an outage.	
Front Door	Managed TLS/SSL certificate is deleted.	Can happen due to mismatch in configuration when deploying. Should be caught in testing stages. Technically the site would still work, but TLS/SSL cert errors will prevent users from accessing it. Mitigation: Re-issuing the cert can take around 20 minutes, plus fixing and re-running the pipeline..	Full
Azure Cosmos DB	Database/collection is renamed.	Can happen due to mismatch in configuration when deploying – Terraform would overwrite the whole database, which could result in data loss. Data loss can be prevented by using database/collection level locks. Application won't be able to access any data. App configuration needs to be updated and pods restarted.	Yes
Azure Cosmos DB	Regional outage	Azure Mission-Critical has multi-region writes enabled. If there's a failure on a read or write, the client retries the current operation. All future operations are permanently routed to the next region in order of preference. In case the preference list had one entry (or was empty) but the account has other regions available, it will route to the next region in the account list.	No
Azure Cosmos DB	Extensive throttling due to lack of RUs.	Depending on the number of RUs and the load-balancing employed at the Front Door level, certain stamps could run hot on Azure Cosmos DB utilization while other stamps can serve more requests. Mitigation: Better load distribution or more RUs.	No

Service	Risk	Impact/Mitigation/Comment	Outage
Azure Cosmos DB	Partition full	Azure Cosmos DB logical partition size limit is 20 GB. If data for a partition key within a container reaches this size, additional write requests will fail with the error "Partition key reached maximum size".	Partial (DB writes disabled)
Azure Container Registry	Regional outage	Container registry uses Traffic Manager to fail over between replica regions. Any request should be automatically rerouted to another region. At worst, Docker images can't be pulled for a few minutes by AKS nodes while DNS failover happens.	No
Azure Container Registry	Image(s) deleted	No images can be pulled. This outage should only affect newly spawned/rebooted nodes. Existing nodes should have the images cached. Mitigation: If detected quickly rerunning the latest build pipelines should bring the images back into the registry.	No
Azure Container Registry	Throttling	Throttling can delay scale-out operations which can result in a temporarily degraded performance. Mitigation: Azure Premium SKU which provides 10k read operations per minute. Container images are optimized and have only small numbers of layers. ImagePullPolicy is set to IfNotPresent to use locally cached versions first. Comment: Pulling a container image consists of multiple read operations, depending on the number of layers. The number of read operations per minute is limited and depends on the ACR SKU size .	No
Azure Kubernetes	Cluster upgrade fails	AKS Node upgrades should occur at different times across the	No

Service	Risk	Impact/Mitigation/Comment	Outage
Service		stamps. if one upgrade fails, the other cluster shouldn't be affected. Cluster upgrades should deploy in a rolling fashion across the nodes to prevent all nodes from becoming unavailable.	
Azure Kubernetes Service	Application pod is killed when serving request.	This could result in end user facing errors and poor user experience. <i>Mitigation:</i> Kubernetes by default removes pods in a graceful way. Pods are removed from services first and the workload receives a SIGTERM with a grace period to finish open requests and write data before terminating. The application code needs to understand SIGTERM and the grace period might need to be adjusted if the workload takes longer to shutdown.	No
Azure Kubernetes Service	Compute capacity unavailable in region to add more nodes.	Scale up/out operations will fail, but it shouldn't affect existing nodes and their operation. Ideally traffic should shift automatically to other regions for load balancing.	No
Azure Kubernetes Service	Subscription reaches CPU core quota to add new nodes.	Scale up/out operations will fail, but it shouldn't affect existing nodes and their operation. Ideally traffic should shift automatically to other regions for load balancing.	No
Azure Kubernetes Service	Let's Encrypt TLS/SSL certificates can't be issued/renewed.	Cluster should report unhealthy towards Front Door and traffic should shift to other stamps. Mitigation: Investigate root cause of issue/renew failure.	No
Azure Kubernetes Service	When resource requests/limits are configured incorrectly, pods can reach 100% CPU utilization and fail requests. Application retry	No (if load not excessive)	

Service	Risk	Impact/Mitigation/Comment	Outage
	mechanism should be able to recover failed requests. Retries could cause a longer request duration, without surfacing the error to the client. Excessive load will eventually cause failure.		
Azure Kubernetes Service	3rd-party container images / registry unavailable	Some components like cert-manager and ingress-nginx require downloading container images and helm charts from external container registries (outbound traffic). In case one or more of these repositories or images are unavailable, new instances on new nodes (where the image is not already cached) might not be able to start. <i>Possible mitigation:</i> In some scenarios it could make sense to import 3rd-party container images into the per-solution container registry. This adds additional complexity and should be planned and operationalized carefully.	Partially (during scale and update/upgrade operations)
Event Hub	Messages can't be sent to the Event Hubs	Stamp becomes unusable for write operations. Health service should automatically detect this and take the stamp out of rotation.	No
Event Hub	Messages can't be read by the BackgroundProcessor	Messages will queue up. Messages shouldn't get lost since they're persisted. Currently, this failure isn't covered by the Health Service. There should be monitoring/alerting in place on the worker to detect errors in reading messages. Mitigation: The stamp should be manually disabled until the problem is fixed.	No
Storage account	Storage account becomes unusable by the worker for	Stamp won't process messages from the Event Hubs. The storage	No

Service	Risk	Impact/Mitigation/Comment	Outage
	Event Hubs check pointing	account is also used by the HealthService. It's expected issues with storage should be detected by the HealthService and the stamp should be taken out of rotation. It can be expected that other services in the stamp will be affected concurrently.	
Storage account	Static website encounters issues.	If serving of the static web site encounters issues, this failure should be detected by Front Door. Traffic won't be sent to this storage account. Caching at Front Door can also alleviate this issue.	No
Key Vault	Key Vault unavailable for <code>GetSecret</code> operations.	At the start of new pods, the AKS CSI driver will fetch all secrets from Key Vault and fail. Pods will be unable to start. There's an automatic update currently every 5 minutes. The update will fail. Errors should show up in <code>kubectl describe pod</code> but the pod keeps working.	No
Key Vault	Key Vault unavailable for <code>GetSecret</code> or <code>SetSecret</code> operations.	New deployments can't be executed. Currently, this failure might cause the entire deployment pipeline to stop, even if only one region is affected.	No
Key Vault	Key Vault throttling	Key Vault has a limit of 1000 operations per 10 seconds. Because of the automatic update of secrets, you could in theory hit this limit if you had many (thousands) of pods in a stamp. Possible mitigation: Decrease update frequency even further or turn it off completely.	No
Application	Misconfiguration	Incorrect connection strings or secrets injected to the app. Should be mitigated by automated deployment (pipeline handles configuration)	No

Service	Risk	Impact/Mitigation/Comment	Outage
		automatically) and blue-green rollout of updates.	
Application	Expired credentials (stamp resource)	<p>If for example, the event hub SAS token or storage account key was changed without properly updating them in Key Vault so that the pods can use them, the respective application component will fail. This failure should then affect the Health Service, and the stamp should be taken out of rotation automatically.</p> <p>Mitigation: Use Microsoft Entra ID-based authentication for all services which support it. AKS requires pods to authenticate using Microsoft Entra Workload ID (preview). Use of Pod Identity was considered in the reference implementation. It was found that Pod Identity wasn't stable enough currently and was decided against use for the current reference architecture. Pod identity could be a solution in the future.</p>	No
Application	Expired credentials (globally shared resource)	<p>If for example, the Azure Cosmos DB API key was changed without properly updating it in all stamp Key Vaults so that the pods can use them, the respective application components will fail. This failure would bring all stamps down at same time and cause a workload-wide outage. For a possible way around the need for keys and secrets using Microsoft Entra auth, see the previous item.</p>	Full
Virtual network	Subnet IP address space exhausted	<p>If the IP address space on a subnet is exhausted, no scale out operations, such as creating new AKS nodes or pods, can happen. It will not lead to an outage but might decrease performance and impact user experience.</p>	No

Service	Risk	Impact/Mitigation/Comment	Outage
		Mitigation: increase the IP space (if possible). If that is not an option, it might help to increase the resources per Node (larger VM SKUs) or per pod (more CPU/memory), so that each pod can handle more traffic, thus decreasing the need for scale out.	

Next steps

Deploy the reference implementation to get a full understanding of the resources and their configuration used in this architecture.

Implementation: Mission-Critical Online

Security considerations for mission-critical workloads

Article • 10/13/2022

Mission-critical workloads are inherently required to be secured - if an application, or its infrastructure, is compromised, availability is at risk. The focus of this architecture is to maximize reliability so that the application remains performant and available under all circumstances. Security controls are applied primarily with the purposes of mitigating threats that impact availability and reliability.

ⓘ Note

Your business requirements might call for more security measures. We highly recommend that you extend the controls in your implementation as per the guidance provided in [Mission-critical guidance in Well-Architected Framework: Security](#).

Identity and access management

At the application level, this architecture uses a simple authentication scheme based on API keys for some restricted operations, such as creating catalog items or deleting comments. Advanced scenarios such as user authentication and user roles are beyond the scope of the [baseline architecture](#).

If your application requires user authentication and account management, follow the principles [outlined in Microsoft Well-Architected Framework](#). Some strategies include, using managed identity providers, avoiding custom identity management, using passwordless authentication whenever possible, and so on.

Least privilege access

Configure access policies such that users and applications get the minimal level of access needed to fulfill their function. Developers typically don't need access to the production infrastructure, but the deployment pipeline needs full access. Kubernetes clusters don't push container images into a registry, but GitHub workflows might. Frontend APIs don't usually get messages from the message broker and backend workers don't necessarily send new messages to the broker. Those decisions depend on

the workload and each component's functionality should be reflected when deciding the access level that should be assigned.

Examples from the Azure Mission-critical reference implementation:

- Each application component that works with Event Hubs uses a connection string with either *Listen* (`BackgroundProcessor`), or *Send* (`CatalogService`) permissions.
That access level ensures that **every pod has only the minimum access required to fulfill its function.**
- The service principal for AKS agent pool has only *Get* and *List* permissions for *Secrets* in Key Vault, no more.
- The AKS Kubelet identity has only the *AcrPull* permission to access the global Container Registry.

Managed identities

To improve the security of a mission-critical workload, where possible, avoid using service-based secrets, such as connection strings or API keys. Using **managed identities** is preferred if the Azure service supports that capability.

The reference implementation uses service-assigned [managed identity](#) of the AKS agent pool ("Kubelet identity") to access the global Azure Container Registry and stamp's Azure Key Vault. Appropriate built-in roles are used to restrict access. For example, this Terraform code assigns only the `AcrPull` role the Kubelet identity:

Terraform

```
resource "azurerm_role_assignment" "acrpull_role" {
    scope              = data.azurerm_container_registry.global.id
    role_definition_name = "AcrPull"
    principal_id      =
azurerm_kubernetes_cluster.stamp.kubelet_identity.0.object_id
}
```

Secrets

Each deployment stamp has its dedicated instance of Azure Key Vault. Until the *Azure AD Workload Identity* feature is available, some parts of the workload use **keys** to access Azure resources, such as Azure Cosmos DB. Those keys are created automatically during deployment and stored in Key Vault with Terraform. **No human operator can interact with secrets, except developers in e2e environments.** In addition, Key Vault access policies are configured in a way that **no user accounts are permitted to access secrets.**

ⓘ Note

This workload doesn't use custom certificates, but the same principles would apply.

On the AKS cluster, the [Azure Key Vault Provider for Secrets Store](#) is used in order for the application to consume secrets. The CSI driver loads keys from Azure Key Vault and mounts them into individual pods' as files.

yml

```
#  
# /src/config/csi-secrets-driver/chart/csi-secrets-driver-  
config/templates/csi-secrets-driver.yaml  
#  
apiVersion: secrets-store.csi.x-k8s.io/v1  
kind: SecretProviderClass  
metadata:  
  name: azure-kv  
spec:  
  provider: azure  
  parameters:  
    usePodIdentity: "false"  
    useVMManagedIdentity: "true"  
    userAssignedIdentityID: {{ .Values.azure.managedIdentityClientId | quote }}  
  }  
  keyvaultName: {{ .Values.azure.keyVaultName | quote }}  
  tenantId: {{ .Values.azure.tenantId | quote }}  
  objects: |  
    array:  
      {{- range .Values.kvSecrets }}  
      - |  
        objectName: {{ . | quote }}  
        objectAlias: {{ . | lower | replace "-" "_" | quote }}  
        objectType: secret  
      {{- end }}  
    
```

The reference implementation uses Helm with Azure Pipelines to deploy the CSI driver containing all key names from Azure Key Vault. The driver is also responsible to refresh mounted secrets if they change in Key Vault.

On the consumer end, both .NET applications use the built-in capability to read configuration from files ([AddKeyPerFile](#)):

C#

```
//  
// /src/app/AlwaysOn.BackgroundProcessor/Program.cs  
// + using Microsoft.Extensions.Configuration;  
//
```

```

public static IHostBuilder CreateHostBuilder(string[] args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureAppConfiguration((context, config) =>
    {
        // Load values from k8s CSI Key Vault driver mount point.
        config.AddKeyPerFile(directoryPath: "/mnt/secrets-store/", optional:
true, reloadOnChange: true);

        // More configuration if needed...
    })
    .ConfigureWebHostDefaults(webBuilder =>
{
    webBuilder.UseStartup<Startup>();
});

```

The combination of CSI driver's auto reload and `reloadOnChange: true` ensures that when keys change in Key Vault, new values are mounted on the cluster. **This doesn't guarantee secret rotation in the application.** The implementation uses singleton Azure Cosmos DB client instance that requires the pod to restart to apply the change.

Custom domains and TLS

Web-based workload should use HTTPS to prevent man-in-the-middle attacks on all interaction levels. For example, communication from the client to API, API to API. Certificate rotation should be automated because expired certificates are still a common cause of outages, or degraded experiences.

The reference implementation fully supports HTTPS with custom domain names, for example, `contoso.com`, and applies appropriate configuration to the `int` and `prod` environments. For `e2e` environments, custom domains can also be added, however, it was decided not to use custom domain names in the reference implementation due to the short-lived nature of `e2e` and the increased deployment time when using custom domains with SSL certificates in Front Door.

To enable full automation of the deployment, the custom domain is expected to be managed through an **Azure DNS Zone**. Infrastructure deployment pipeline dynamically creates CNAME records in the Azure DNS zone and maps these records automatically to an Azure Front Door instance.

Front Door-managed SSL certificates are enabled, which removes the need for manual SSL certificate renewals. **TLS 1.2** is configured as the minimum version.

```

#
# /src/infra/workload/globalresources/frontdoor.tf
#
resource "azurerm_frontdoor_custom_https_configuration"
"custom_domain_https" {
    count                      = var.custom_fqdn != "" ? 1 : 0
    frontend_endpoint_id       =
"${azurerm_frontdoor.main.id}/frontendEndpoints/${local.frontdoor_custom_fro
ntend_name}"
    custom_https_provisioning_enabled = true

    custom_https_configuration {
        certificate_source = "FrontDoor"
    }
}

```

Environments that aren't provisioned with custom domains can be accessed through the default Front Door endpoint, for example `env123.azurefd.net`.

Note

On the cluster ingress controller, custom domains aren't used in either case. Instead an Azure-provided DNS name such as `[prefix]-cluster.[region].cloudapp.azure.com` is used with Let's Encrypt enabled to issue free SSL certificates for those endpoints.

The reference implementation uses Jetstack's `cert-manager` to auto-provision SSL/TLS certificates (from Let's Encrypt) for ingress rules. More configuration settings like the `ClusterIssuer` (used to request certificates from Let's Encrypt) are deployed via a separate `cert-manager-config` helm chart stored in [src/config/cert-manager/chart](#).

This implementation is using `ClusterIssuer` instead of `Issuer` as documented [here](#) and [here](#) to avoid having issuers per namespaces.

YAML

```

apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging
spec:
  acme:

```

Configuration

All application runtime configuration is stored in Azure Key Vault including secrets and non-sensitive settings. A configuration store, such as Azure App Configuration, could be used to store the settings, however, having a single store reduces the number of potential points of failure for mission-critical applications. Using Key Vault for runtime configuration simplifies the overall implementation.

Key Vaults should be populated by the deployment pipeline. In the implementation, the required values are either sourced directly from Terraform, such as database connection strings, or passed through as Terraform variables from the deployment pipeline.

Infrastructure and deployment configuration of individual environments (`e2e`, `int`, `prod`) is stored in variable files that are part of the source code repository. This approach has two benefits:

- All changes in an environment are tracked and go through deployment pipelines before they're applied to the environment.
- Individual e2e environments can be configured differently, because deployment is based on code in a branch.

One exception is the storage of **sensitive values** for pipelines. These values are stored as secrets in Azure DevOps variable groups.

Container security

Securing container images is necessary for all containerized workloads.

Workload Docker containers used in the reference implementation are based on **runtime images**, not SDK, to minimize footprint and potential attack surface. There are no additional tools installed (such as `ping`, `wget`, or `curl`).

The application runs under a non-privileged user `workload`, created as part of the image build process:

Dockerfile

```
RUN groupadd -r workload && useradd --no-log-init -r -g workload workload
USER workload
```

The reference implementation uses Helm to package the YAML manifests needed to deploy individual components together, including their Kubernetes deployment, services, auto-scaling (HPA) configuration, and security context. All Helm charts contain foundational security measures following Kubernetes best practices.

These security measures are:

- `readOnlyFilesystem`: The root filesystem `/` in each container is set to **read-only** to prevent the container from writing to the host filesystem. This restriction prevents attackers from downloading more tools and persisting code in the container. Directories that require read-write access are mounted as volumes.
- `privileged`: All containers are set to run as **non-privileged**. Running a container as privileged gives all capabilities to the container, and it also lifts all the limitations enforced by the device control group controller.
- `allowPrivilegeEscalation`: Prevents the inside of a container from gaining more privileges than its parent process.

These security measures are also configured for 3rd-party containers and Helm charts (that is, `cert-manager`) when possible and audited by Azure Policy.

YAML

```
#  
# Example:  
# /src/app/charts/backgroundprocessor/values.yaml  
#  
containerSecurityContext:  
    privileged: false  
    readOnlyRootFilesystem: true  
    allowPrivilegeEscalation: false
```

Each environment (*prod*, *int*, every *e2e*) has a **dedicated instance of Azure Container Registry**, with global replication to each of the regions where stamps are deployed.

① Note

Current reference implementation doesn't use vulnerability scanning of Docker images. It is recommended to utilize **Microsoft Defender for container registries**, potentially with **GitHub Actions**.

Traffic ingress

Azure Front Door is the global load balancer in this architecture. All web requests are routed through Front Door, which selects the appropriate backend. Mission-critical application should take advantage of other capabilities of Front Door, such as Web Application Firewall (WAF).

Web Application Firewall

An important Front Door capability is the **Web Application Firewall (WAF)**, because Front Door is able to inspect traffic, which is passing through. In the **Prevention** mode, all suspicious requests are blocked. In the implementation, two rulesets are configured: `Microsoft_DefaultRuleSet` and `Microsoft_BotManagerRuleSet`.

💡 Tip

When deploying Front Door with WAF it's recommended that you start with the **Detection** mode, closely monitor its behavior with natural end-user traffic, and fine-tune the detection rules. After false positives are eliminated, or rare, switch to **Prevention** mode. This is necessary, because every application is different and some payloads can be considered malicious, while completely legitimate for that particular workload.

Routing

Only those requests that come through Azure Front Door will be routed to the API containers (`CatalogService` and `HealthService`). This behavior is enforced by using an Nginx ingress configuration, which checks the `X-Azure-FDID` header - not just the presence of it, but also if it's the right one for the global Front Door instance of a particular environment.

```
yml
```

```
#  
# /src/app/charts/catalogservice/templates/ingress.yaml  
#  
apiVersion: networking.k8s.io/v1  
kind: Ingress  
metadata:  
  # ...  
  annotations:  
    # To restrict traffic coming only through our Front Door instance, we use  
    # a header check on the X-Azure-FDID  
    # The value gets injected by the pipeline. Hence, this ID should be  
    # treated as a sensitive value  
    nginx.ingress.kubernetes.io/modsecurity-snippet: |  
      SecRuleEngine On  
      SecRule &REQUEST_HEADERS:X-Azure-FDID "@eq 0\"  
      \"log,deny,id:106,status:403,msg:'Front Door ID not present'\"  
      SecRule REQUEST_HEADERS:X-Azure-FDID "@rx ^(?!{{  
        .Values.azure.frontdoorid }}).*\$\"  \"log,deny,id:107,status:403,msg:'Wrong
```

```
Front Door ID\\"
# ...
```

Deployment pipelines ensure that this header is properly populated, but there's also a need to **bypass this restriction for smoke tests**, because they probe each cluster directly, not through Front Door. The reference implementation uses the fact that smoke tests are triggered as part of the deployment and therefore the header value is known and can be added to smoke test HTTP requests:

PowerShell

```
# / .ado/pipelines/scripts/Run-SmokeTests.ps1
#
$header = @{
    "X-Azure-FDID" = "$frontdoorHeaderId"
    "X-TEST-DATA" = "true" # Header to indicate that posted comments and
rating are just for test and can be deleted again by the app
}
```

Secure deployments

Following the baseline well-architected principles for operational excellence, **all deployments should be fully automated** and there shouldn't be manual steps required except triggering the run, or approving a gate.

Malicious attempts or accidental misconfiguration that can disable security measures must be prevented. The reference implementation uses the same pipeline for both infrastructure and application deployment, which forces an **automated rollback of any potential configuration drift**, maintaining the integrity of the infrastructure and alignment with the application code. Any change is discarded on the next deploy.

Sensitive values for deployment are either generated during the pipeline run (by Terraform), or supplied as Azure DevOps secrets. These values are protected with role-based access restrictions.

ⓘ Note

GitHub workflows offer similar concept [↗](#) of separate store for secret values. Secrets are encrypted environmental variables that can be used by GitHub Actions.

It's important to **pay attention to any artifacts produced by the pipeline**, because those can potentially contain secret values or information about inner workings of the

application. The Azure DevOps deployment of the reference implementation generates two files with Terraform outputs: one for stamp and one for global infrastructure. These files don't contain passwords, which would allow direct compromise of the infrastructure. However they can be considered semi-sensitive, because they reveal information about the infrastructure - cluster IDs, IP addresses, Storage Account names, Key Vault names, Azure Cosmos DB database name, Front Door header ID, and others.

For workloads that utilize **Terraform**, extra effort needs to be put into **protecting the state file**, as it contains full deployment context, **including secrets**. The state file is typically stored in a Storage Account that should have a separate lifecycle from the workload and should be accessible only from a deployment pipeline. Any other access to this file should be logged and alerts sent to appropriate security group.

Dependency updates

Libraries, frameworks and tools used by the application get updated over time and it's important to follow these updates regularly, because they often contain security fixes, which could allow attackers unauthorized access into the system.

The reference implementation uses GitHub's **Dependabot** for NuGet, Docker, npm, Terraform and GitHub Actions dependency updates. The `dependabot.yml` configuration file is automatically generated with a PowerShell script, because of the complexity of the various parts of the application (for example, each Terraform module needs a separate entry).

```
yml

#
# /.github/dependabot.yml
#
version: 2
updates:
- package-ecosystem: "nuget"
  directory: "/src/app/AlwaysOn.HealthService"
  schedule:
    interval: "monthly"
  target-branch: "component-updates"

- package-ecosystem: "docker"
  directory: "/src/app/AlwaysOn.HealthService"
  schedule:
    interval: "monthly"
  target-branch: "component-updates"

# ... the rest of the file...
```

- **Updates are triggered monthly** as a compromise between having the most up-to-date libraries and keeping the overhead maintainable. Additionally, key tools (Terraform) are monitored continuously and important updates are executed manually.
- **Pull requests** are targeting the `component-updates` branch, instead of `main`.
- **npm libraries** are configured to check only dependencies that go to the compiled application, not the supporting tools like `@vue-cli`.

Dependabot creates a separate pull request (PR) for each update, which can get overwhelming for the operations team. The reference implementation first collects a batch of updates in the `component-updates` branch, then runs tests in the `e2e` environment and if successful, another PR is created into the `main` branch.

Defensive coding

API calls can fail due to various reasons, code errors, malfunctioned deployments, infrastructure failures, or others. In that case, the caller (client application) shouldn't receive extensive debugging information, because that could provide adversaries with helpful data points about the application.

The reference implementation demonstrates this principle by **returning only the correlation ID** in the failed response and doesn't share the failure reason (like exception message or stack trace). Using this ID (and with the help of `X-Server-Location` header) an operator is able to investigate the incident using Application Insights

C#

```
//  
// Example ASP.NET Core middleware which adds the Correlation ID to every  
API response.  
  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    // ...  
  
    app.Use(async (context, next) =>  
    {  
        context.Response.OnStarting(o =>  
        {  
            if (o is HttpContext ctx)  
            {  
                context.Response.Headers.Add("X-Server-Name",  
Environment.MachineName);  
                context.Response.Headers.Add("X-Server-Location",  
sysConfig.AzureRegion);  
                context.Response.Headers.Add("X-Correlation-ID",  

```

```
Activity.Current?.RootId);
    context.Response.Headers.Add("X-Requested-Api-Version",
ctx.GetRequestedApiVersion()?.ToString());
}
return Task.CompletedTask;
}, context);
await next();
});

// ...
}
```

Next

Deploy the reference implementation to get a full understanding of resources and their configuration.

[Implementation: Mission-Critical Online](#)

Operations for mission-critical workloads on Azure

Article • 11/30/2023

Like with any application, change will occur in your mission-critical workloads. The application will evolve over time, keys will expire, patches will be released, and more. All changes and maintenance should be applied using deployment pipelines. This article provides operational guidance for making common changes and updates.

Organizational alignment is equally important to operation procedures. It's crucial for the operational success of a mission-critical workload that the end-to-end responsibilities fall within a single team, the DevOps team.

The technical execution should take advantage of Azure-native platform capabilities, and the use of automated Azure Pipelines to deploy changes to the application, infrastructure, and configuration. Again, maintenance tasks should be automated and manual tasks should be avoided.

The following sections describe approaches to handling different types of change.

Application automation

Continuous Integration and Continuous Deployment (CI/CD) enables the proper deployment and verification of mission-critical workloads. CI/CD is the preferred approach to deploy changes to any environment, Dev/Test, production, and others. For mission-critical workloads, the changes listed below should result in the deployment of an entirely new stamp. The new stamp should be thoroughly tested as part of the release process before traffic is routed to the stamp via a blue/green deployment strategy.

The following sections describe changes that should be implemented, where possible, through CI/CD.

Application changes

All changes to the application code should be deployed through CI/CD. The code should be built, linted, and tested against regressions. Application dependencies, such as runtime environment or packages should be monitored, with updates deployed via CI/CD.

Infrastructure changes

Infrastructure should be modeled and provisioned as code. This practice is commonly referred to as Infrastructure as Code (IaC). All changes to the IaC should be deployed through the CI/CD pipelines. Updates to the infrastructure, such as patching the OS should also be managed via CI/CD pipelines.

Configuration changes

Configuration changes are a common cause of application outages. To combat these outages, configuration for application or infrastructure should be captured as code. This practice is known as Configuration as Code (CaC). Changes to CaC should be deployed via CI/CD pipelines.

Library/SDK updates

For mission-critical applications, it's critical that source code and dependencies are updated when new versions become available. The recommended approach is to take advantage of configuration management change process in the source code repository. It should be configured to automatically create Pull Requests for various dependency updates, such as:

- .NET NuGet packages
- JavaScript Node Package Manager packages
- Terraform Provider

The following is an example of automating library updates using [dependabot](#) in a GitHub repository.

1. Dependabot detects updates of libraries and SDK used in application code
2. Dependabot updates the application code in a branch and creates a pull request (PR) with those changes against the main branch. The PR contains all relevant

information and is ready for final review.

The screenshot shows a GitHub pull request page. At the top, it says "Bump Microsoft.Azure.Cosmos from 3.30.0 to 3.30.1 in /src/app/AlwaysOn.BackgroundProcessor #581". A green button on the right says "New issue". Below this, a purple button says "Merged". A message indicates "merged 1 commit into component-updates from dependabot/nuget/src/app/AlwaysOn.BackgroundProcessor/component-updates/Microsoft.Azure.Cosmos-3.30.1" 16 minutes ago. The pull request interface includes tabs for Conversation (0), Commits (1), Checks (0), Files changed (1), and a status bar showing +1 -1. The main content area shows a comment from "dependabot (bot)" that reads: "Bumps Microsoft.Azure.Cosmos from 3.30.0 to 3.30.1." It links to Release notes, Changelog, and Commits. Below this, there's a note about compatibility: "compatibility unknown". It states that Dependabot will resolve conflicts if not altered manually by commenting "@dependabot rebase". There's also a section for Dependabot commands and options. At the bottom, it shows a commit from "dependabot (bot)" requesting a review from a code owner 20 hours ago. The commit hash is f912f88. On the right side, there are sections for Reviewers, Assignees (no one assigned), Labels (dependencies, .NET), Projects (None yet), Milestone (No milestone), and Development (a note about successfully merging may close issues). A circular icon with a magnifying glass is shown next to the development note.

3. When code review and testing are done, the PR can be merged to the main branch.

For dependencies dependabot isn't able to monitor, ensure that you have processes in place to detect new releases.

Key/Secret/Certificate rotations

Rotating (renewing) keys and secrets should be a standard procedure in any workload. Secrets might need to be changed on short notice after being exposed or regularly as a good security practice.

Because expired or invalid secrets can cause outages to the application ([see Failure Analysis](#)), it's important to have a clearly defined and proven process in place. For Azure Mission-Critical, stamps are only expected to live for a few weeks. Because of that, rotating secrets of stamp resources isn't a concern. If secrets in one stamp expire, the application as a whole would continue to function.

Management of secrets to access long-living global resources, however, are critical. A notable example is the Azure Cosmos DB API keys. If Azure Cosmos DB API keys expire, all stamps will be affected simultaneously and cause a complete outage of the application.

The following is Azure mission critical tested and documented approach for rotating Azure Cosmos DB keys without causing downtime to services running in Azure Kubernetes Service.

1. Update stamps with secondary key. By default, the primary API key for Azure Cosmos DB is stored as a secret in Azure Key Vault in each stamp. Create a PR that updates the IaC template code to use the secondary Azure Cosmos DB API key. Run this change through the normal PR review and update procedure to get deployed as a new release or as a hotfix.
2. (optional) If the update was deployed as a hotfix to an existing release, the pods will automatically pick up the new secret from Azure Key Vault after a few minutes. However, the Azure Cosmos DB client code does currently not reinitialize with a changed key. To resolve this issue, restart all pods manually using the following commands on the clusters:

```
Bash
```

```
kubectl rollout restart deployment/CatalogService-deploy -n workload  
kubectl rollout restart deployment/BackgroundProcessor-deploy -n workload  
kubectl rollout restart deployment/healthservice-deploy -n workload
```

3. Newly deployed or restarted pods will now use the secondary API key for the connection to Azure Cosmos DB.
4. Once all pods on all stamps are restarted, or a new stamp has been deployed, regenerate the primary API key for Azure Cosmos DB. Here's an example for the command:

```
Bash
```

```
az cosmosdb keys regenerate --key-kind primary --name MyCosmosDBDatabaseAccount --resource-group MyResourceGroup
```

5. Change the IaC template back to use the primary API key for future deployments. Alternatively, you can continue to use the secondary key and switch back to the primary API key when the time comes to renew the secondary.

Alerts

Alerts are key to understanding if and when there are issues with your environment. Changes to alerts and/or action groups should be implemented via CI/CD pipelines. For more information on alerts, see [Health modeling and observability of mission-critical workloads on Azure](#).

Automation

Many platforms and services running on Azure provide automation for common operational activities. This automation includes autoscaling and the automated handling of keys and certificates.

Scaling

As part of the application design, the scale requirements that define a scale-unit for the stamp as a whole should be determined. The individual services within the stamp need to be able to scale out to meet peak demand or scale in to save money or resources.

Services that don't have enough resources can exhibit different side effects, including the following:

- An insufficient number of pods processing messages from a queue/topic/partition will result in a growing number of unprocessed messages. This is sometimes referred to as a growing queue depth.
- Insufficient resources on an AKS node can result in pods not being able to run.
- The following will result in throttled requests:
 - Insufficient Request Units (RUs) for Azure Cosmos DB
 - Insufficient processing units (PUs) for Event Hubs premium or throughput units (TUs) for standard
 - Insufficient Messaging Units (MUs) for Service Bus premium tier

Take advantage of autoscaling features of the services, where possible, to ensure you have enough resources to meet demand. The following are automatic scaling features you can take advantage of:

- [Horizontal Pod Autoscaling](#) allows you to increase or decrease the number of pods running workloads, depending upon demand.
- The [AKS cluster autoscaler](#) allows you to increase or decrease the number of nodes in the cluster, depending upon demand.
- You can [automatically scale up Azure Event Hubs throughput units \(standard tier\)](#)
- You can [Automatically update messaging units of an Azure Service Bus namespace](#)

Managing keys, secrets, and certificates

Use managed identities where possible to avoid having to manage API keys or secrets such as passwords.

When you're using keys, secrets, or certificates, use Azure-native platform capabilities whenever possible. The following are some examples of these platform-level capabilities:

- Azure Front Door has built-in capabilities for TLS certificate management and renewal.
- Azure Key Vault supports automatic key rotation.

Manual operations

There are operational activities that require manual intervention. These processes should be tested.

Dead-lettered messages

Messages that can't be processed should be routed to a dead-letter queue with an alert configured for that queue. These messages usually require manual intervention to understand and mitigate the issue. You should build the ability to view, update and replay dead-lettered messages.

Azure Cosmos DB restore

When Azure Cosmos DB data is unintentionally deleted, updated, or corrupted, you need to perform a restore from a periodic backup. Restoring from a periodic backup can only be accomplished via a support case. This process should be documented and periodically tested.

Quota increases

Azure subscriptions have quota limits. Deployments can fail when these limits are reached. Some quotas are adjustable. For adjustable quotas, you can request an increase from the [My quotas](#) page on the Azure portal. For non-adjustable quotas you, need to submit a support request. The Azure support team will work with you to find a solution.

Important

See [Operational procedures for mission-critical workloads on Azure](#) for operational design considerations and recommendations.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal authors:

- [Rob Bagby](#) | Principal Content Developer
- [Allen Sudbring](#) | Senior Content Developer

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

Deploy the reference implementation to get a full understanding of the resources and their configuration used in this architecture.

[Implementation: Mission-Critical Online](#)

Continuous validation with Azure Load Testing and Azure Chaos Studio

Article • 11/29/2023

As cloud-native applications and services become more complex, deploying changes and new releases for them can be challenging. Outages are frequently caused by faulty deployments or releases. But **errors can also occur after deployment**, when an application starts receiving real traffic, especially in complex workloads that run in highly distributed multitenant cloud environments and that are maintained by multiple development teams. These environments require more resiliency measures, like retry logic and autoscaling, which are usually hard to test during the development process.

That's why **continuous validation in an environment that's similar to the production environment is important**, so that you can find and fix any problems or bugs as early in the development cycle as possible. Workload teams should test early in the development process (shift left) and make it convenient for developers to do testing in an environment that's close to the production environment.

Mission-critical workloads have high availability requirements, with targets of 3, 4, or 5 nines (99.9%, 99.99%, or 99.999%, respectively). It's crucial to implement **rigorous automated testing** to reach those goals.

Continuous validation depends on each workload and on architectural characteristics. This article provides a guide for preparing and integrating Azure Load Testing and Azure Chaos Studio into a regular development cycle.

1 – Define tests based on expected thresholds

Continuous testing is a complex process that requires proper preparation. What will be tested and the expected outcomes must be clear.

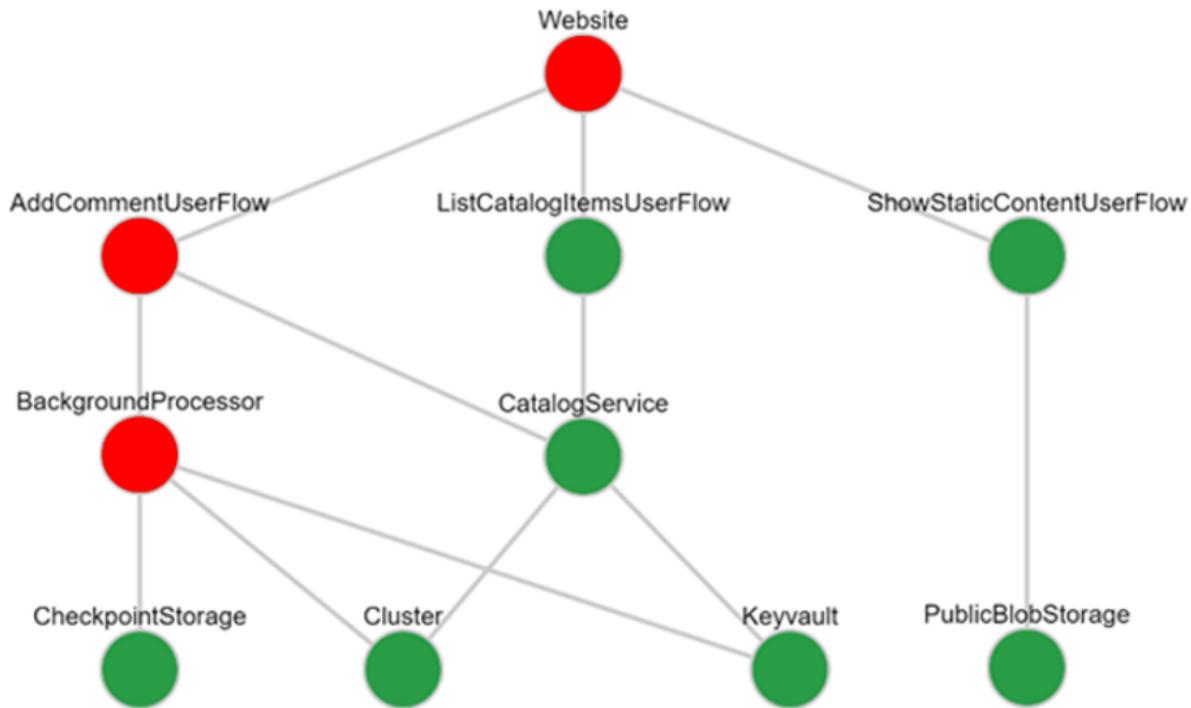
In [PE:06 - Recommendations for performance testing](#) and [RE:08 - Recommendations for designing a reliability testing strategy](#), the Azure Well-Architected Framework recommends that you start by **identifying key scenarios, dependencies, expected usage, availability, performance, and scalability targets**.

You should then define a set of **measurable threshold values** to quantify the expected performance of the key scenarios.

💡 Tip

Examples of threshold values include the expected number of user sign-ins, requests per second for a given API, and operations per second for a background process.

You should use threshold values to develop a [health model for your application](#), both for testing and for operating the application in production.



Next, use the values to define a **load test** that generates realistic traffic for testing application baseline performance, validating expected scale operations, and so on. Sustained artificial user traffic is needed in pre-production environments, because without usage it's difficult to reveal runtime issues.

Load testing ensures that changes made to the application or infrastructure don't cause issues and the system still meets the expected performance and test criteria. A failed test run that doesn't meet the test criteria indicates that you need to adjust the baseline, or that an unexpected error occurred.

Load test results		Engine health	
Statistics			
Load 1737617 Total requests	Duration 5 mins, 59 secs	Response time 491.00 ms 90th percentile response time	Error percentage 61.30 % Aggregate requests which failed
Throughput 4853.68 /s Request rate			
Test criteria			
Metric	Aggregate function	Condition	Threshold
Requests per second	Average	Less than	1200
Response time	Average	Greater than	75
Error	Percentage	Greater than	50
		Request name	Actual value
			Result
			Passed
			Failed
			Failed

Even though automated tests represent day-to-day usage, you should run manual load tests regularly to verify how the system responds to unexpected peaks.

The second part of continuous validation is the **injection of failures** (chaos engineering). This step verifies the resiliency of a system by testing how it responds to faults. Also, that all the resiliency measures, such as retry logic, autoscaling, and others, are working as expected.

2 - Implement validation with Load Testing and Chaos Studio

Microsoft Azure provides these managed services to implement load testing and chaos engineering:

- [Azure Load Testing](#) produces synthetic user load on applications and services.
- [Azure Chaos Studio](#) provides the ability to perform chaos experimentation, by systematically injecting failures into application components and infrastructure.

You can deploy and configure both Chaos Studio and Load Testing via the Azure portal, but, in the context of continuous validation, it's more important that you have APIs to deploy, configure, and run tests in a **programmatic and automated way**. Using these two tools together enables you to observe how the system reacts to problems and its ability to self-heal in response to infrastructure or application failures.

The following video shows a [combined implementation of Chaos and Load Testing integrated in Azure DevOps](#):

[https://www.microsoft.com/en-us/videoplayer/embed/RE4Y50k?postJs||Msg=true ↗](https://www.microsoft.com/en-us/videoplayer/embed/RE4Y50k?postJs||Msg=true)

If you're developing a mission-critical workload, take advantage of the reference architectures, detailed guidance, sample implementations, and code artifacts provided as part of the [Azure Mission-Critical project](#) ↗ and [Azure Well-Architected Framework](#).

The Mission-Critical implementation deploys the Load Testing service via Terraform and contains a [collection of PowerShell Core wrapper scripts](#) ↗ to interact with the service via its API. These scripts can be embedded directly into a deployment pipeline.

One option in the reference implementation is to execute the load test directly from within the end-to-end (e2e) pipeline that is used to spin up individual (branch specific) development environments:

Run pipeline

X

Select parameters below and manually run the pipeline

Branch/tag

main



Select a branch from the list or enter the name of a tag as refs/tags/<tagname>

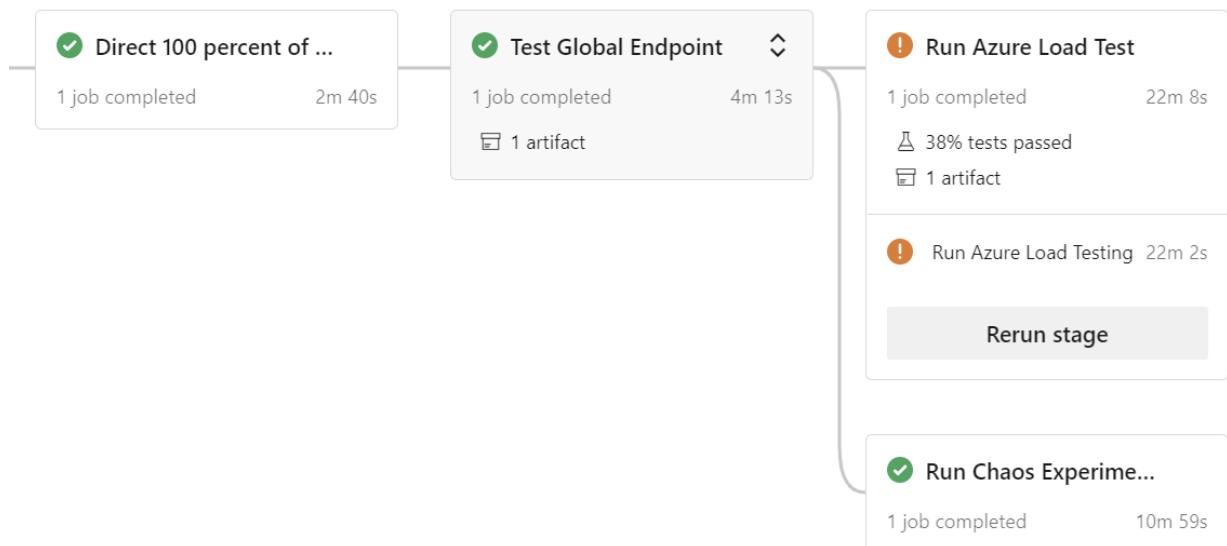
Commit

Run chaos testing

Run load testing

Destroy environment at the end

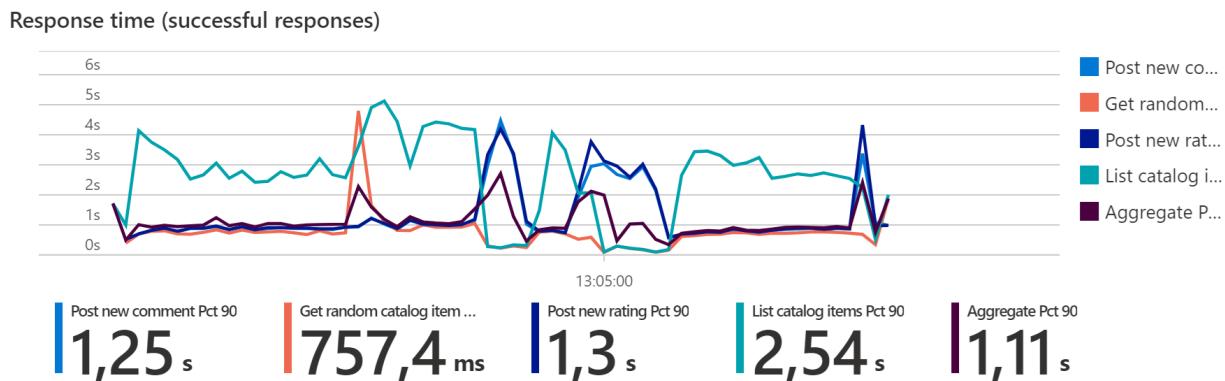
The pipeline will automatically run a load test, with or without chaos experiments (depending on the selection) in parallel:



ⓘ Note

Running chaos experiments during a load test can result in higher latency, higher response times and temporarily increased error rates. You'll notice higher numbers

until a scale-out operation completes or a failover has completed, when compared to a run without chaos experiments.



Depending on whether chaos testing is enabled and the choice of experiments, baseline definitions might vary, because the tolerance for errors can be different in "normal" state and "chaos" state.

3 – Adjust thresholds and establish a baseline

Finally, **adjust the load test thresholds** for regular runs to verify that the application (still) provides the expected performance and doesn't produce any errors. Have a separate baseline for chaos testing that tolerates expected spikes in error rates and temporary reduced performance. This activity is continuous and needs to be repeated regularly. For example, after introducing new features, changing service SKUs, and others.

The Azure Load Testing service provides a built-in capability called **test criteria** that allows specifying certain criteria that a test needs to pass. This capability can be used to implement different baselines.

Test criteria						
Metric	Aggregate function	Condition	Threshold	Actual value	Result	
Requests per second	Average	Less than	1200	4840	✓ Passed	
Response time	Average	Greater than	75	184.6	✗ Failed	
Error	Percentage	Greater than	50	61.3	✗ Failed	

The capability is available through the Azure portal, and via the load testing API, and the wrapper scripts developed as part of Azure Mission-critical provide an option to handover a JSON-based baseline definition.

We highly recommend **integrating these tests directly into your CI/CD pipelines** and running them during the early stages of feature development. For an example, see the [sample implementation](#) in the Azure Mission-critical reference implementation.

In summary, failure is inevitable in any complex distributed system and the solution must therefore be architected (and tested) to handle failures. The [Well-Architected Framework mission-critical workload guidance](#) and reference implementations can help you design and operate highly reliable applications to derive maximum value from the Microsoft cloud.

Next step

Review the deployment and testing design area for mission-critical workloads.

[Design area: Deployment and testing](#)

Related resources

- [Azure Load Testing documentation](#)
- [Azure Chaos Studio documentation](#)

Global routing redundancy for mission-critical web applications

Article • 04/19/2023

ⓘ Important

Designing redundancy implementations that deal with global platform outages for a mission-critical architecture can be complex and costly. Because of the potential problems that might arise with this design, carefully consider the [tradeoffs](#).

In most situations, you won't need the architecture described in this article.

Mission-critical systems strive to minimize single points of failure by building redundancy and self-healing capabilities in the solution as much as possible. Any unified entry point of the system can be considered a point of failure. If this component experiences an outage, the entire system will be offline to the user. When choosing a routing service, it's important to consider the reliability of the service itself.

In the [baseline architecture for a mission-critical application](#), Azure Front Door was chosen because of its high uptime Service-level agreements (SLA) and a rich feature set:

- Route traffic to multiple regions in an active-active model
- Transparent failover using TCP anycast
- Serve static content from edge nodes by using integrated content delivery networks (CDNs)
- Block unauthorized access with integrated web application firewall

Front Door is designed to provide the utmost resiliency and availability for not only our external customers, but also for multiple properties across Microsoft. For more information about Front Door's capabilities, see [Accelerate and secure your web application with Azure Front Door](#).

Front Door capabilities are more than enough to meet most business requirements, however, with any distributed system, expect failure. If the business requirements demand a higher composite SLA or zero-down time in case of an outage, you'll need to rely on an alternate traffic ingress path. However, the pursuit of a higher SLA comes with significant costs, operational overhead, and can lower your overall reliability. Carefully consider the [tradeoffs](#) and potential issues that the alternate path might introduce in other components that are on the critical path. Even when the impact of unavailability is significant, complexity might outweigh the benefit.

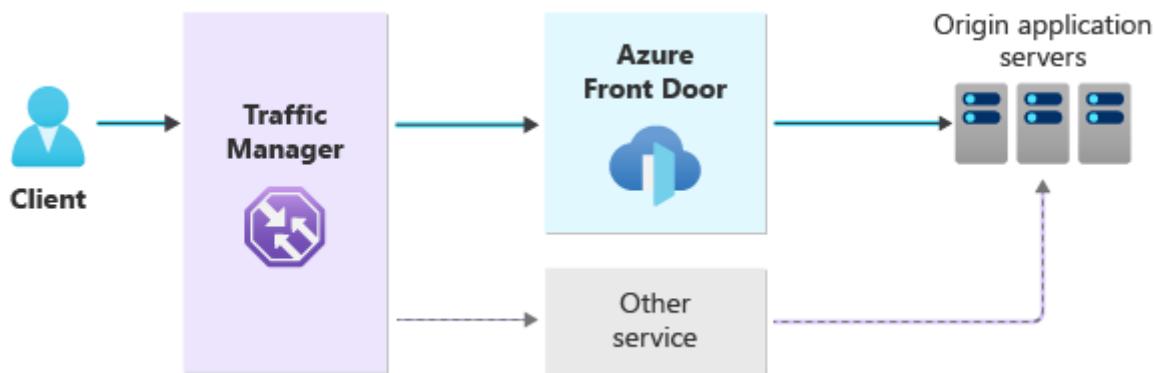
One approach is to define a secondary path, with alternate service(s), which becomes active only when Azure Front Door is unavailable. Feature parity with Front Door shouldn't be treated as a hard requirement. Prioritize features that you absolutely need for business continuity purposes, even potentially running in a limited capacity.

Another approach is using third-party technology for global routing. This approach will require a multicloud active-active deployment with stamps hosted across two or more cloud providers. Even though Azure can effectively be integrated with other cloud platforms, this approach isn't recommended because of operational complexity across the different cloud platforms.

This article describes some strategies for global routing using Azure Traffic Manager as the alternate router in situations where Azure Front Door isn't available.

Approach

This architecture diagram shows a general approach with multiple redundant traffic paths.



With this approach, we will introduce several components and provide guidance that will make significant changes associated to the delivery of your web application(s):

1. [Azure Traffic Manager](#) directs traffic to Azure Front Door or to the alternative service that you've selected.

Azure Traffic Manager is a DNS-based global load balancer. Your domain's CNAME record points to Traffic Manager, which determines the destination based on how you configure its [routing method](#). Using [priority routing](#) will make traffic flow through Azure Front Door by default. Traffic Manager can automatically switch traffic to your alternate path if Azure Front Door is unavailable.

Important

This solution mitigates risks associated with Azure Front Door outages, but it's susceptible to Azure Traffic Manager outages as a global point of failure.

You can also consider using a different global traffic routing system, such as a global load balancer. However, Traffic Manager works well for many situations.

2. You have two ingress paths:

- Azure Front Door provides the primary path and processes and routes all of your application traffic.
- Another router is used as a backup for Azure Front Door. Traffic only flows through this secondary path if Front Door is unavailable.

The specific service that you select for the secondary router depends on many factors. You might choose to use Azure-native services, or third-party services. In these articles we provide Azure-native options to avoid adding additional operational complexity to the solution. If you use third-party services, you need to use multiple control planes to manage your solution.

3. Your origin application servers need to be ready to accept traffic from either service. Consider how you [secure traffic to your origin](#), and what responsibilities Front Door and other upstream services provide. Ensure that your application can handle traffic from whichever path your traffic flows through.

Tradeoffs

While this mitigation strategy can make the application be available during platform outages, there are some significant tradeoffs. You should weigh the potential benefits against known costs, and make an informed decision about whether the benefits are worth those costs.

- **Financial cost:** When you deploy multiple redundant paths to your application, you need to consider the cost of deploying and running the resources. We provide two example scenarios for different use cases, each of which has a different cost profile.
- **Operational complexity:** Every time you add additional components to your solution, you increase your management overhead. Any change to one component might impact other components.

Suppose you decide to use the new capabilities of Azure Front Door. You need to check whether your alternative traffic path also provides an equivalent capability, and if not, you need to decide how to handle the difference in behavior between

the two traffic paths. In real-world applications, these complexities can have a high cost, and can present a major risk to your system's stability.

- **Performance:** This design requires additional CNAME lookups during name resolution. In most applications, this isn't a significant concern, but you should evaluate whether your application performance is affected by introducing additional layers into your ingress path.
- **Opportunity cost:** Designing and implementing redundant ingress paths requires a significant engineering investment, which ultimately comes at an opportunity cost to feature development and other platform improvements.

Warning

If you're not careful in how you design and implement a complex high-availability solution, you can actually make your availability worse. Increasing the number of components in your architecture increases the number of failure points. It also means you have a higher level of operational complexity. When you add extra components, every change that you make needs to be carefully reviewed to understand how it affects your overall solution.

Availability of Azure Traffic Manager

Azure Traffic Manager is a reliable service, but the service level agreement doesn't guarantee 100% availability. If Traffic Manager is unavailable, your users might not be able to access your application, even if Azure Front Door and your alternative service are both available. It's important to plan how your solution will continue to operate under these circumstances.

Traffic Manager returns cacheable DNS responses. If time to live (TTL) on your DNS records allows caching, short outages of Traffic Manager might not be a concern. That is because downstream DNS resolvers might have cached a previous response. You should plan for prolonged outages. You might choose to manually reconfigure your DNS servers to direct users to Azure Front Door if Traffic Manager is unavailable.

Traffic routing consistency

It's important to understand the Azure Front Door capabilities and features that you use and rely on. When you choose the alternate service, decide the minimum capabilities that you need and omit other features when your solution is in a degraded mode.

When planning an alternative traffic path, here are some key questions you should consider:

- Do you use Azure Front Door's caching features? If caching is unavailable, can your origin servers keep up with your traffic?
- Do you do use the Azure Front Door rules engine to perform custom routing logic, or to rewrite requests?
- Do you use the Azure Front Door web application firewall (WAF) to secure your applications?
- Do you restrict traffic based on IP address or geography?
- Who issues and managed your TLS certificates?
- How do you restrict access to your origin application servers to ensure it comes through Azure Front Door? Do you use Private Link, or do you rely on public IP addresses with service tags and identifier headers?
- Do your application servers accept traffic from anywhere other than Azure Front Door? If they do, which protocols do they accept?
- Do your clients use Azure Front Door's HTTP/2 support?

Web application firewall (WAF)

If you use Azure Front Door's WAF to protect your application, consider what happens if the traffic doesn't go through Azure Front Door.

If your alternative path also provides a WAF, consider the following questions:

- Can it be configured in the same way as your Azure Front Door WAF?
- Does it need to be tuned and tested independently, to reduce the likelihood of false positive detections?

Warning

You might choose not to use WAF for your alternative ingress path. This approach can be considered to support the reliability target of the application. However, this isn't a good practice and we don't recommend it.

Consider the tradeoff in accepting traffic from the internet without any checks. If an attacker discovers an unprotected secondary traffic path to your application, they might send malicious traffic through your secondary path even when the primary path includes a WAF.

It's best to **secure all paths** to your application servers.

Domain names and DNS

Your mission-critical application should use a custom domain name. You'll control over how traffic flows to your application, and you reduce the dependencies on a single provider.

It's also a good practice to use a high-quality and resilient DNS service for your domain name, such as [Azure DNS](#). If your domain name's DNS servers are unavailable, users can't reach your service.

It's recommended that you use multiple DNS resolvers to increase overall resiliency even further.

CNAME chaining

Solutions that combine Traffic Manager, Azure Front Door, and other services use a multi-layer DNS CNAME resolution process, also called CNAME chaining. For example, when you resolve your own custom domain, you might see five or more CNAME records before an IP address is returned.

Adding additional links to a CNAME chain can affect DNS name resolution performance. However, DNS responses are usually cached, which reduces the performance impact.

TLS certificates

For a mission-critical application, it's recommended that you provision and use your own TLS certificates instead of the managed certificates provided by Azure Front Door. You'll reduce the number of potential problems with this complex architecture.

Here are some benefits:

- To issue and renew managed TLS certificates, Azure Front Door verifies your ownership of the domain. The domain verification process assumes that your domain's CNAME records point directly to Azure Front Door. But, that assumption often isn't correct. Issuing and renewing managed TLS certificates on Azure Front Door might not work smoothly and you increase the risk of outages due to TLS certificate problems.
- Even if your other services provide managed TLS certificates, they might not be able to verify domain ownership.
- If each service gets their own managed TLS certificates independently, there might be issues. For example, users might not expect to see different TLS certificates

issued by different authorities, or with different expiry dates or thumbprints.

However, there will be additional operations related to renewing and updating your certificates before they expire.

Origin security

When you [configure your origin](#) to only accept traffic through Azure Front Door, you gain protection against layer 3 and layer 4 [DDoS attacks](#). Because Azure Front Door only responds to valid HTTP traffic, it also helps to reduce your exposure to many protocol-based threats. If you change your architecture to allow alternative ingress paths, you need to evaluate whether you've accidentally increased your origin's exposure to threats.

If you use Private Link to connect from Azure Front Door to your origin server, how does traffic flow through your alternative path? Can you use private IP addresses to access your origins, or must you use public IP addresses?

If your origin uses the Azure Front Door service tag and the X-Azure-FDID header to validate that traffic has flowed through Azure Front Door, consider how your origins can be reconfigured to validate that traffic has flowed through either of your valid paths. You must test that you haven't accidentally opened your origin to traffic through other paths, including from other customers' Azure Front Door profiles.

When you plan your origin security, check whether the alternative traffic path relies on provisioning dedicated public IP addresses. This might need a manually triggered process to bring the backup path online.

If there are dedicated public IP addresses, consider whether you should implement [Azure DDoS Protection](#) to reduce the risk of denial of service attacks against your origins. Also, consider whether you need to implement [Azure Firewall](#) or another firewall capable of protecting you against a variety of network threats. You might also need more intrusion detection strategies. These controls can be important elements in a more complex multi-path architecture.

Health modeling

Mission-critical design methodology requires a system [health model](#) that gives you overall observability of the solution and its components. When you use multiple traffic ingress paths, you need to monitor the health of each path. If your traffic is rerouted to the secondary ingress path, your health model should reflect the fact that the system is still operational but that it's running in a degraded state.

Include these questions in your health model design:

- How do the different components of your solution monitor the health of downstream components?
- When should health monitors consider downstream components to be unhealthy?
- How long does it take for an outage to be detected?
- After an outage is detected, how long does it take for traffic to be routed through an alternative path?

There are multiple global load balancing solutions that enable you to monitor the health of Azure Front Door and trigger an automatic failover to a backup platform if an outage occurs. Azure Traffic Manager is suitable in most cases. With Traffic Manager, you configure [endpoint monitoring](#) to monitor downstream services by specifying which URL to check, how frequently to check that URL, and when to consider the downstream service unhealthy based on probe responses. In general, the shorter the interval between checks, the less time it takes for Traffic Manager to direct traffic through an alternative path to reach your origin server.

If Front Door is unavailable, then multiple factors influence the amount of time that the outage affects your traffic, including:

- The time to live (TTL) on your DNS records.
- How frequently Traffic Manager runs its health checks.
- How many failed probes Traffic Manager is configured to see before it reroutes traffic.
- How long clients and upstream DNS servers cache Traffic Manager's DNS responses for.

You also need to determine which of those factors are within your control and whether upstream services beyond your control might affect user experience. For example, even if you use low TTL on your DNS records, upstream DNS caches might still serve stale responses for longer than they should. This behavior might exacerbate the effects of an outage or make it seem like your application is unavailable, even when Traffic Manager has already switched to sending requests to the alternative traffic path.

Tip

Mission-critical solutions require automated failover approaches wherever possible. Manual failover processes are considered slow in order for the application to remain responsive.

Refer to: [Mission-critical design area: Health modeling](#)

Zero-downtime deployment

When you're planning how to operate a solution with redundant ingress path, you should also plan for how you deploy or configure your services when they're degraded. For most Azure services, SLAs apply to the uptime of the service itself, and not to management operations or deployments. Consider whether your deployment and configuration processes need to be made resilient to service outages.

You should also consider the number of independent control planes that you need to interact with to manage your solution. When you use Azure services, Azure Resource Manager provides a unified and consistent control plane. However, if you use a third-party service to route traffic, you might need to use a separate control plane to configure the service, which introduces further operational complexity.

Warning

The use of multiple control planes introduces complexity and risk to your solution. Every point of difference increases the likelihood that somebody accidentally misses a configuration setting, or apply different configurations to redundant components. Ensure that your operational procedures mitigate this risk.

Refer to: [Mission-critical design area: Zero-downtime deployment](#)

Continuous validation

For a mission-critical solution, your testing practices need to verify that your solution meets your requirements regardless of the path that your application traffic flows through. Consider each part of the solution and how you test it for each type of outage.

Ensure that your testing processes include these elements:

- Can you verify that traffic is correctly redirected through the alternative path when the primary path is unavailable?
- Can both paths support the level of production traffic you expect to receive?
- Are both paths adequately secured, to avoid opening or exposing vulnerabilities when you're in a degraded state?

Refer to: [Mission-critical design area: Continuous validation](#)

Common scenarios

Here are common scenarios where this design can be used:

- [Global content delivery](#) commonly applies to static content delivery, media, and high-scale eCommerce applications. In this scenario, caching is a critical part of the solution architecture, and failures to cache can result in significantly degraded performance or reliability.
- [Global HTTP ingress](#) commonly applies to mission-critical dynamic applications and APIs. In this scenario, the core requirement is to route traffic to the origin server reliably and efficiently. Frequently, a WAF is an important security control used in these solutions.

Warning

If you're not careful in how you design and implement a complex multi-ingress solution, you can actually make your availability worse. Increasing the number of components in your architecture increases the number of failure points. It also means you have a higher level of operational complexity. When you add extra components, every change that you make needs to be carefully reviewed to understand how it affects your overall solution.

Next steps

Review the [global HTTP ingress](#) and [global content delivery](#) scenarios to understand whether they apply to your solution.

Mission-critical global content delivery

Article • 04/19/2023

Caching is a common way to reduce load on the backend services and optimize performance for users. Content delivery networks (CDNs), including Azure Front Door, provide caching at the network edge.

Mission-critical workloads often use multiple CDNs to achieve a higher level of uptime. If one CDN experiences outage or degraded performance, your traffic is automatically diverted to another CDN.

If you implement multiple CDNs, consider the implications of this approach. Each CDN provides a separate network path to your application servers, and you need to configure and test each CDN separately.

This article describes an approach for using Azure Front Door with a partner CDN, Verizon. This approach is suitable for solutions that rely heavily on caching for delivering static content delivery, media, and high-scale eCommerce applications.

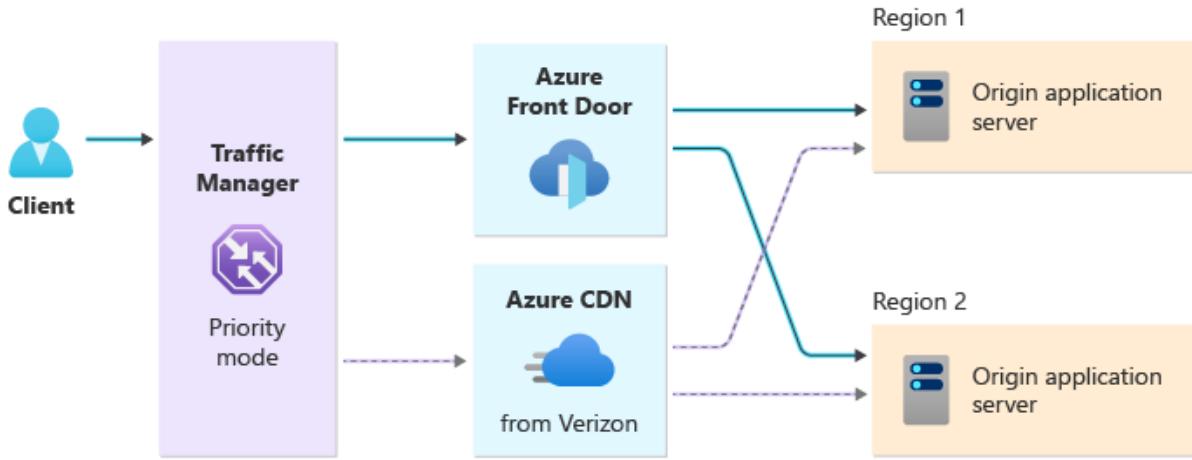
Note

This use case is part of an overall design strategy that covers an alternate approach when Azure Front Door is unavailable. For information about the context and considerations, see [Mission-critical global web applications](#).

Approach

Verizon's CDN and the CDN platform (Edgio) can be integrated into your Azure solution. You can configure it from the Azure portal and APIs. The platform is isolated from Microsoft's infrastructure.

This isolation provides a high degree of resiliency from disaster scenarios. If an outage or disaster occurs, traffic is automatically shifted between Azure Front Door and Verizon's CDN. You can use Azure Traffic Manager to detect an outage and redirect traffic to the alternative CDN.



- **Traffic Manager using priority routing mode** has two [endpoints](#). By default, Traffic Manager sends requests through Azure Front Door. If Azure Front Door is unavailable, Traffic Manager sends the request through the partner CDN instead.
- **Azure Front Door** processes and routes most of your application traffic. Azure Front Door routes traffic to the appropriate origin application server, and it provides the primary path to your application. If Azure Front Door is unavailable, traffic is automatically redirected through the secondary path.
- **Azure CDN from Verizon** is configured to send traffic to each origin server.
- **Your origin application servers** need to be ready to accept traffic from both Azure Front Door and Azure CDN from Verizon, at any time.

Considerations

The considerations described in [Mission-critical global web applications](#) still apply to this use case. Here are some additional points:

Choice of CDN

In this example, we suggest using Verizon's CDN. Verizon's CDN is often a good choice because it can be deployed, configured, and billed through Azure, reducing your operational complexity. It runs on separate physical infrastructure to Azure Front Door, which means it's resilient to outages or problems on Microsoft's infrastructure.

You might choose to use a different CDN, or even to use multiple CDNs, depending on your requirements and risk tolerance.

Feature parity

Azure Front Door and Verizon's CDN provide distinct capabilities, and features aren't equivalent between the two products. For example, there are differences in handling of TLS certificates, WAF, and HTTP rules.

Carefully consider the features of Azure Front Door that you use, and whether your alternative CDN has equivalent capabilities. For more information, see [Consistency of ingress paths](#).

Cache fill

If you're running multiple CDNs in active-passive mode, during a failover, CDN configured in passive mode needs to perform a *cache fill* from your origin during a failover.

Test the failover between Azure Front Door and your alternative CDN to detect anomalies or performance issues.

If your solution is at risk from performance issues during cache fills, consider these approaches to reduce the risk:

- **Scale out or scale** up your origins to cope with higher traffic levels, especially during a cache fill.
- **Prefill both CDNs.** You serve a percentage of your most popular content through the passive CDN even before a failover event occurs. For example, you could consider using [weighted traffic routing mode](#).

Tradeoffs

Using multiple CDNs comes with some tradeoffs.

- **Cost.** There might be an increase in the overall cost of the solution. When you deploy a multi-CDN architecture, you're billed for multiple CDNs. Make sure that you understand how you're charged for each CDN in your solution, and all of the other components you deploy.
- **Performance.** There might be performance issues during failover between Azure Front Door and your alternative CDN.

A common issue is [cache refilling](#) when CDNs are running in an active-passive mode. The CDN configured in passive mode needs refill its cache from the origin. It can overload origin systems during that process.

Next steps

Review the [global HTTP ingress](#) scenario to understand whether it applies to your solution.

Mission-critical global HTTP ingress

Article • 04/19/2023

Mission-critical applications need to maintain a high level of uptime, even when network components are unavailable or degraded. When you design web traffic ingress, routing, and security, you can consider combining multiple Azure services to achieve a higher level of availability and to avoid having a single point of failure.

If you decide to adopt this approach, you'll need to implement separate network path to your application servers, and each path needs to be configured and tested separately. You must carefully consider the full implications of this approach.

This article describes an approach to support global HTTP traffic ingress through Azure Front Door and Azure Application Gateway. This approach might suit your needs if your solution needs:

- Azure Front Door for global traffic routing. This might mean that you have multiple instances of your application in separate Azure regions, or that you serve all global users from a single region.
- Web application firewall (WAF) to protect your application, regardless of the path your traffic follows to reach your origin servers.

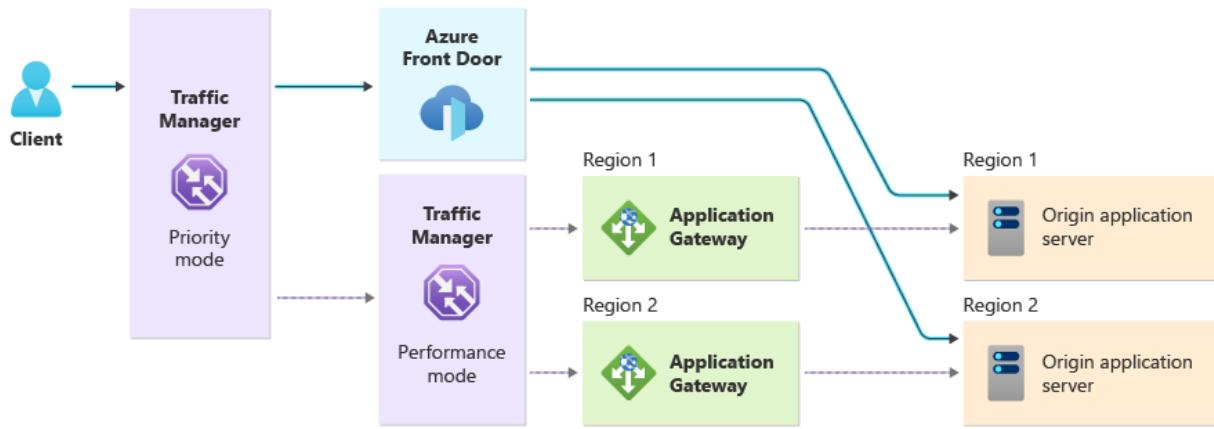
Caching at the network edge isn't critical part of your application delivery. If caching is important, see [Mission-critical global content delivery](#) for an alternative approach.

Note

This use case is part of an overall design strategy that covers an alternate approach when Azure Front Door is unavailable. For information about the context and considerations, see [Mission-critical global web applications](#).

Approach

This DNS-based load balancing solution uses multiple Azure Traffic Manager profiles to monitor Azure Front Door. In the unlikely event of an availability issue, Traffic Manager redirects traffic through Application Gateway.



The solution includes the following components:

- **Traffic Manager using priority routing mode** has two [endpoints](#). By default, Traffic Manager sends requests through Azure Front Door. If Azure Front Door is unavailable, a second Traffic Manager profile determines where to direct the request. The second profile is described below.
- **Azure Front Door** processes and routes most of your application traffic. Azure Front Door routes traffic to the appropriate origin application server, and it provides the primary path to your application. Azure Front Door's WAF protects your application against common security threats. If Azure Front Door is unavailable, traffic is automatically redirected through the secondary path.
- **Traffic Manager using performance routing mode** has an endpoint for each Application Gateway instance. This Traffic Manager sends requests to the Application Gateway instance with the best performance from the client's location.
- **Application Gateway** is deployed into each region, and sends traffic to the origin servers within that region. Application Gateway's WAF protects any traffic that flows through the secondary path.
- **Your origin application servers** need to be ready to accept traffic from both Azure Front Door and Azure Application Gateway, at any time.

Considerations

The following sections describe some important considerations for this type of architecture. You should also review [Mission-critical global web applications](#) for other important considerations and tradeoffs when you use Azure Front Door in a mission-critical solution.

Traffic Manager configuration

This approach uses [nested Traffic Manager profiles](#) to achieve both priority-based and performance-based routing together for your application's alternative traffic path. In a simple scenario with an origin in a single region, you might only need a single Traffic Manager profile configured to use priority-based routing.

Regional distribution

Azure Front Door is a global service, while Application Gateway is a regional service.

Azure Front Door's points of presence are deployed globally, and TCP and TLS connections [terminate at the closest point of presence to the client](#). This behavior improves the performance of your application. In contrast, when clients connect to Application Gateway, their TCP and TLS connections terminate at the Application Gateway that receives the request, regardless of where the traffic originated.

Connections from clients

As a global multitenant service, Azure Front Door provides inherent protection against a variety of threats. Azure Front Door only accepts valid HTTP and HTTPS traffic, and doesn't accept traffic on other protocols. Microsoft manages the public IP addresses that Azure Front Door uses for its inbound connections. Because of these characteristics, Azure Front Door can help to [protect your origin against various attack types](#), and your origins can be [configured to use Private Link connectivity](#).

In contrast, Application Gateway is an internet-facing service with a dedicated public IP address. You must protect your network and origin servers against a variety of attack types. For more information, see [Origin security](#).

Private Link connections to origin servers

Azure Front Door Premium provides [Private Link connectivity](#) to your origins, which reduces the public internet-facing surface area of your solution.

If you use Private Link to connect to your origins, consider deploying a private endpoint into your virtual network, and configure Application Gateway to use the private endpoint as the backend for your application.

Scaling

When you deploy Application Gateway, you deploy dedicated compute resources for your solution. If large amounts of traffic arrive at your Application Gateway

unexpectedly, you might observe performance or reliability issues.

To mitigate this risk, consider how you [scale your Application Gateway instance](#). Either use autoscaling, or ensure that you've manually scaled it to handle the amount of traffic that you might receive after failing over.

Caching

If you use Azure Front Door's caching features, then it's important to be aware that after your traffic switches to the alternative path and uses Application Gateway, content is no longer served from the Azure Front Door caches.

If you depend on caching for your solution, see [Mission-critical global content delivery](#) for an alternative approach that uses a partner CDN as a fallback to Azure Front Door.

Alternatively, if you use caching but it's not an essential part of your application delivery strategy, consider whether you can scale out or scale up your origins to cope with the increased load that was caused by the higher number of cache misses during a failover.

Tradeoffs

This type of architecture is most useful if you want your alternative traffic path to use features like request processing rules, a WAF, and TLS offload. Both Azure Front Door and Application Gateway provide similar capabilities.

However, there are tradeoffs:

- **Operational complexity.** When you use any of these features, you need to configure them on both Azure Front Door and Application Gateway. For example, if you make a configuration change to your Azure Front Door WAF, you need to apply the same configuration change to your Application Gateway WAF too. Your operational complexity becomes much higher when you need to reconfigure and test two separate systems.
- **Feature parity.** While there are similarities between the features that Azure Front Door and Application Gateway offer, many features don't have exact parity. Be mindful of these differences, because they could affect how the application is delivered based on the traffic path it follows.

Application Gateway doesn't provide caching. For more information about this difference, see [Caching](#).

Azure Front Door and Application Gateway are distinct products and have different use cases. In particular, [the two products are different in how they're deployed to Azure regions](#). Ensure you understand the details of each product and how you use them.

- **Cost.** You typically need to deploy an Application Gateway instance into each region where you have an origin. Because each Application Gateway instance is billed separately, the cost can become high when you have origins deployed into several regions.

If cost is a significant factor for your solution, see [Mission-critical global content delivery](#) for an alternative approach that uses a partner content delivery network (CDN) as a fallback to Azure Front Door. Some CDNs bill for traffic on a consumption basis, so this approach might be more cost-effective. However, you might lose some of the other advantages of using Application Gateway for your solution.

Alternatively, you could consider deploying an alternative architecture where Traffic Manager can route traffic directly to platform as a service (PaaS) application services, avoiding the need for Application Gateway and reducing your costs. You could consider this approach if you use a service like Azure App Service or Azure Container Apps for your solution. However, if you follow this approach, there are several important tradeoffs to consider:

- **WAF:** Azure Front Door and Application Gateway both provide WAF capabilities. If you expose your application services directly to the internet, you might not be able to protect your application with a WAF.
- **TLS offload:** Azure Front Door and Application Gateway both terminate TLS connections. Your application services need to be configured to terminate TLS connections.
- **Routing:** Both Azure Front Door and Application Gateway perform routing across multiple origins or backends, including path-based routing, and they support complex routing rules. If your application services are exposed directly to the internet, you can't perform traffic routing.

Warning

If you consider exposing your application directly to the internet, create a thorough threat model and ensure that the architecture meets your security, performance, and resiliency requirements.

If you use virtual machines to host your solution, you should not expose the virtual machines to the internet.

Next steps

Review the [global content delivery](#) scenario to understand whether it applies to your solution.

Microsoft Cloud for Developers

Learn how to integrate services across the Microsoft Cloud with tutorials, videos, code samples, documentation and more.



GET STARTED
[Build applications on the Microsoft Cloud](#)



HOW - TO GUIDE
[GitHub Copilot ↗](#)



OVERVIEW
[Azure OpenAI Service Prompt Engineering](#)



OVERVIEW
[Dev Proxy](#)

Microsoft Cloud Learning Resources



Copilot

Adopt, extend and build Copilot experiences
ISV Extensibility options for Copilot
Copilot for Dynamics 365
Copilot for Microsoft 365
Copilot in Power Apps
[GitHub Copilot ↗](#)



Solutions Across the Microsoft Cloud

[Azure and Dynamics 365 Scenarios](#)
[Azure and Microsoft 365 Scenarios](#)
[Azure and Power Platform Scenarios](#)



Hands-on Tutorials



Code Samples

[Audio/Video Calling from a Custom App into a Teams Meeting](#)

[Automate Data Reporting with Azure Functions and Power Automate ↗](#)

[Build Productivity Apps by using Microsoft Graph Toolkit](#)

[Integrate OpenAI, Communication, and Organizational Data Features into a Line of Business App](#)

[Transform your Business Applications with Fusion Development](#)

[Audio/Video Calling from a Custom App into a Teams Meeting ↗](#)

[Building Real-Time Collaborative Apps ↗](#)

[Integrate AI, Communication, and Organizational Data Features into a Line of Business App ↗](#)

[Microsoft Teams App Camp ↗](#)

The Microsoft Cloud

Azure

- [!\[\]\(38f3c34eb987129a7733a3a6036e3ccd_img.jpg\) Create an Azure Account ↗](#)
- [!\[\]\(80afb3a825a13da3812843ded5eee096_img.jpg\) Azure Portal ↗](#)
- [!\[\]\(95f2c0c48e0f15f66713b5e8bd2262dc_img.jpg\) Azure Services](#)
- [!\[\]\(8d92ac41d4f27eda26aecfddf6e7d228_img.jpg\) Azure Architecture Center](#)
- [!\[\]\(72882a2ac2da9b2f59611364c3333144_img.jpg\) Azure Well-Architected Framework](#)
- [!\[\]\(b5c739e31dfd2aea3b1a82d23ed02781_img.jpg\) Cloud Adoption Framework for Azure](#)

GitHub

- [!\[\]\(57295be7bb9ea247c149dcb1547f539b_img.jpg\) Create a GitHub Account ↗](#)
- [!\[\]\(c0419c7fdb02280c2c8ab2b95a35caf9_img.jpg\) GitHub Documentation ↗](#)
- [!\[\]\(ac17d6b5aa8af45303d367a75ea77370_img.jpg\) GitHub Actions ↗](#)
- [!\[\]\(88d3a3196670cbcace818b735c51d939_img.jpg\) GitHub Copilot ↗](#)

Microsoft 365

- [!\[\]\(7893661e2cbe0f8ce6a11af79bdcfca7_img.jpg\) Join the Microsoft 365 Developer Program](#)
- [!\[\]\(9140af3931a6334f432475f0608e5db2_img.jpg\) Microsoft 365 Developer Documentation](#)
- [!\[\]\(f3295ec0e623997eaba45c1305970b65_img.jpg\) Microsoft Graph Documentation](#)
- [!\[\]\(a5ecf0fccc6ec3617f58ddb27666c823_img.jpg\) Microsoft Teams Developer Documentation](#)
- [!\[\]\(22db78219c441807d7ad22be94498cb3_img.jpg\) SharePoint Developer](#)

Power Platform

- [Create a developer environment](#)
- [Power Platform Developer Documentation](#)
- [Power Platform Developer Training](#)
- [Power Platform Community ↗](#)

Dynamics 365

- [Dynamics 365 Free Trial ↗](#)
- [Dynamics 365 Developer Documentation](#)
- [Dynamics 365 Developer Training](#)
- [Dynamics 365 Community ↗](#)

Industry and Government

- [Get Microsoft Industry Clouds ↗](#)
- [Microsoft Industry Clouds Documentation](#)
- [Browse Industry Clouds ↗](#)
- [Azure Government Free Trial ↗](#)
- [Azure Government Documentation](#)
- [Compare Azure Government and Global Azure](#)

Microsoft Cloud SDKs and Additional Tooling

Software Developer Kits (SDKs) and additional tools to integrate applications with the Microsoft Cloud.

Azure CLI

The Azure Command-Line Interface (CLI) is a cross-platform command-line tool to connect to Azure and execute...

GitHub Codespaces ↗

A codespace is a development environment that's hosted in the cloud.

Visual Studio Family ↗

Download and install Visual Studio, Visual Studio for Mac, or Visual Studio Code

Azure SDKs and Tools ↗

The Azure SDKs are collections of libraries built to make it easier to use Azure services from your language of choice.

CLI for Microsoft 365 ↗

The CLI for Microsoft 365 can help you manage your Microsoft 365 tenant and SharePoint Framework projec...

Dev Proxy

Dev Proxy is a command line tool for testing Microsoft Graph, SharePoint Online and any other HTTP APIs.

[Microsoft Graph SDKs](#)

The Microsoft Graph SDKs are designed to simplify building high-quality, efficient, and resilient applications that...

[Power Platform CLI](#)

Microsoft Power Platform CLI is a simple, one-stop developer CLI that empowers developers and ISVs to perform various...

[PowerShell](#)

PowerShell is a cross-platform task automation solution made up of a command-line shell, a scripting language, and a...

Azure and Power Platform scenarios

Article • 11/01/2023

Power Platform [↗](#) provides tools for analyzing data, building solutions, automating processes, and creating virtual agents. Power Platform includes these products:

- [Power BI](#) [↗](#). Enable your employees to generate data-driven insights.
- [Power Apps](#) [↗](#). Enable anyone to build custom apps.
- [Power Automate](#) [↗](#). Give everyone the ability to automate organizational processes.
- [Power Virtual Agents](#) [↗](#). Build chatbots to engage with your customers and employees—no coding required.

This article provides summaries of solutions and architectures that use Power Platform together with Azure services.

Anyone can be a developer with Power Platform. Check out this short video to learn more:

<https://www.youtube-nocookie.com/embed/2dscy89ks9I> [↗](#)

Apache®, Apache Ignite, Ignite, and the flame logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of these marks.

Solutions across Azure and Power Platform

The following articles provide detailed analysis of solutions that feature integration between Azure and Power Platform.

Power Platform (general)

Architecture	Summary	Technology focus
Citizen AI with Power Platform	Learn how to use Azure Machine Learning and Power Platform to quickly create a machine learning proof of concept and a production version.	AI
Custom business processes	Deploy portals that use Power Apps to automate manual or paper-based processes and provide rich user	Integration

Architecture	Summary	Technology focus
	experiences. Use Power BI to generate reports.	
Modern data warehouse for small and medium businesses	Use Azure Synapse Analytics, Azure SQL Database, and Azure Data Lake Storage to modernize legacy and on-premises data. This solution integrates easily with Power Platform.	Analytics
Virtual health on Microsoft Cloud for Healthcare	Develop a virtual health solution by using Microsoft Cloud for Healthcare. This solution uses Power Apps to host a patient portal and store data, Power BI for reporting, and Power Automate to trigger notifications.	Web

[Browse all our Power Platform solutions.](#)

Power Apps

Architecture	Summary	Technology focus
Custom business processes	Deploy portals that use Power Apps to automate manual or paper-based processes and provide rich user experiences.	Integration
CI/CD for Microsoft Power Platform	Learn how to create an Azure CI/CD pipeline to manage your Power Platform application lifecycle.	DevOps
Eventual consistency between multiple Power Apps instances	Handle dependent data in a resilient way in Power Apps.	Web
Line of business extension	Modernize legacy systems by automating processes. Schedule calculations, connect to third-party data sources or legacy systems, and process and share data. Power Apps retrieves the data, and Power BI provides reporting.	Integration
Web and mobile front ends	Accelerate development by using a visual designer. Use Azure Functions for low-latency processing and Power Apps and Power Automate for out-of-the-box connectors.	Integration

[Browse all our Power Apps solutions.](#)

Power Automate

Architecture	Summary	Technology focus
Extract text from objects using Power Automate and AI Builder	Use AI Builder and Azure Form Recognizer in a Power Automate workflow to extract text from images. The text can be used for indexing and retrieval.	AI
Power Automate deployment at scale	Learn how to use a hub-and-spoke architectural model to deploy Power Automate parent and child flows.	Integration
Web and mobile front ends	Accelerate development by using a visual designer. Use Azure Functions for low-latency processing and Power Apps and Power Automate for out-of-the-box connectors.	Integration

[Browse all our Power Automate solutions.](#)

Power BI

Architecture	Summary	Technology focus
Advanced analytics	Combine any data at any scale and then use custom machine learning to get near real-time data analytics on streaming services. Power BI provides querying and reporting.	Analytics
Azure Machine Learning architecture	Learn how to build, deploy, and manage high-quality models with Azure Machine Learning, a service for the end-to-end machine learning lifecycle. Power BI provides data visualization.	AI
Campaign optimization with Azure HDInsight Spark	Use Microsoft Machine Learning Server to build and deploy a machine learning model to maximize the purchase rate of leads that are targeted by a marketing campaign. Power BI provides summaries of the effectiveness of the campaign recommendations.	Databases
Clinical insights with Microsoft Cloud for Healthcare	Gather insights from clinical and medical data by using Microsoft Cloud for Healthcare. Power BI reports provide insights on healthcare metrics.	Web
Data analysis for regulated industries	Learn about an architecture that you can use for data analysis workloads in regulated industries. The architecture includes ETL/ELT and Power BI.	Analytics
Data governance with Profisee and	Integrate Profisee master data management with Azure Purview to build a foundation for data governance and	Databases

Architecture	Summary	Technology focus
Azure Purview	management. Produce and deliver high-quality, trusted data. Power BI is used as an analytics tool.	
Data management across Azure Data Lake with Azure Purview	Use Azure Purview to build a foundation for data governance and management that can produce and deliver high-quality, trusted data. Azure Purview connects natively with Power BI.	Analytics
Deliver highly scalable customer service and ERP applications	Use Azure SQL, Azure Cosmos DB, and Power BI to deliver highly scalable customer service and enterprise resource planning (ERP) applications that work with structured and unstructured data.	Analytics
Demand forecasting for shipping and distribution	Use historical demand data and the Microsoft AI platform to train a demand forecasting model for shipping and distribution solutions. A Power BI dashboard displays the forecasts.	Analytics
Finance management apps using Azure Database for PostgreSQL	Use Azure Database for PostgreSQL to store critical data with improved security and get high-value analytics and insights over aggregated data. Power BI supports native connectivity with PostgreSQL to ingest data for analytics.	Databases
Finance management apps using Azure Database for MySQL	Use Azure Database for MySQL to store critical data with improved security and get high-value analytics and insights over aggregated data. Power BI provides analytics.	Databases
Forecast energy and power demand	Forecast spikes in demand for energy products and services by using Azure Machine Learning and Power BI.	AI
HIPAA-compliant and HITRUST-compliant health data AI	Store, manage, and analyze HIPAA-compliant and HITRUST-compliant health data and medical records with a high level of built-in security. Power BI provides data visualization.	Serverless
Intelligent apps using Azure Database for MySQL	Use Azure Database for MySQL to develop sophisticated machine learning and visualization apps that provide analytics and information that you can act on. Power BI provides visualization and data analysis.	Databases
Intelligent apps using Azure Database for PostgreSQL	Use Azure Database for PostgreSQL to develop sophisticated machine learning and visualization apps that provide analytics and information that you can act on. Power BI provides visualization and data analysis.	Databases
IoT-connected light, power, and internet for emerging markets	Learn how energy provider Veriown uses solar-powered IoT devices with Azure services to provide clean, low-cost	IoT

Architecture	Summary, and internet service to remote customers. Power BI provides reporting.	Technology focus
IoT using Azure Cosmos DB	Scale instantly and elastically to accommodate diverse and unpredictable IoT workloads without sacrificing ingestion or query performance. Use Power BI to analyze warehoused data.	IoT
Line of business extension	Modernize legacy systems by automating processes. Schedule calculations, connect to third-party data sources or legacy systems, and process and share data by using Power BI.	Integration
Loan charge-off prediction with HDInsight Spark	Learn how lending institutions can use Azure HDInsight and machine learning to predict the likelihood of loans getting charged off. Power BI provides a visualization dashboard.	Databases
Loan credit risk and default modeling	Learn how SQL Server 2016 R Services can help lenders issue fewer unprofitable loans by predicting borrower credit risk and default probability. Power BI provides a dashboard to help lenders make decisions based on the predictions.	Databases
Manage data across your Azure SQL estate with Azure Purview	Improve your organization's governance process by using Azure Purview in your Azure SQL estate. Azure Purview connects natively to Power BI.	Analytics
Master data management with Azure and CluedIn	Use CluedIn eventual connectivity data integration to blend data from many siloed data sources and prepare it for analytics and business operations. Power BI helps you to generate insights from the data.	Databases
Master data management with Profisee and Azure Data Factory	Integrate Profisee master data management with Azure Data Factory to get high-quality data for Azure Synapse and other analytics applications. Power BI provides data analysis.	Databases
Medical data storage solutions	Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security. Power BI provides data analysis.	Storage
Modern analytics architecture with Azure Databricks	Create a modern analytics architecture that uses Azure Databricks, Azure Data Lake Storage, Power BI, and other Azure services. Unify data, analytics, and AI workloads at any scale.	Analytics
Optimize marketing with machine	Build a machine learning model with Azure Machine Learning, Azure Synapse Analytics, and Power BI that	AI

Architecture	Summary	Technology focus
Machine learning	optimizes big data marketing campaigns.	
Population health management for healthcare	Use population health management to improve clinical and health outcomes and reduce costs. Track, monitor, and benchmark data by using this process. Power BI provides analytics.	AI
Predict length of stay and patient flow	Predict capacity and patient flow for your healthcare facility so that you can enhance the quality of care and improve operational efficiency. Power BI provides a dashboard to help you make decisions based on the predictions.	AI
Predict the length of stay in hospitals	Predict length of stay for hospital admissions to enhance care quality and operational workload efficiency and reduce re-admissions. Power BI provides data visualization.	Analytics
Predictive insights with vehicle telematics	Learn how car dealerships, manufacturers, and insurance companies can use Azure to gain predictive insights on vehicle health and driving habits. Power BI provides data visualizations for reporting.	AI
Project 15 Open Platform	Use Internet of Things technologies with the Project 15 Open Platform to accelerate innovation in species tracking, ecosystem monitoring, and other areas. Power BI provides visualization.	IoT
Quality assurance	Build a quality assurance system that collects data and improves productivity by identifying potential problems in a manufacturing pipeline before they occur. Use Power BI to visualize real-time operational dashboards.	AI
Serverless computing solution for LOB apps	Build and run customer onboarding applications without managing or maintaining infrastructure. Improve developer productivity with this serverless architecture. Power BI is used to store customer information.	Serverless
Use a demand forecasting model for price optimization	Predict future customer demand and optimize pricing by using big-data and advanced-analytics services from Azure. Use Power BI to monitor the results.	Analytics

[Browse all our Power BI solutions.](#)

Related resources

- Browse all our Power Platform architectures
- Azure and Microsoft 365 scenarios
- Azure and Dynamics 365 scenarios

Build CI/CD with Azure for Microsoft Power Platform

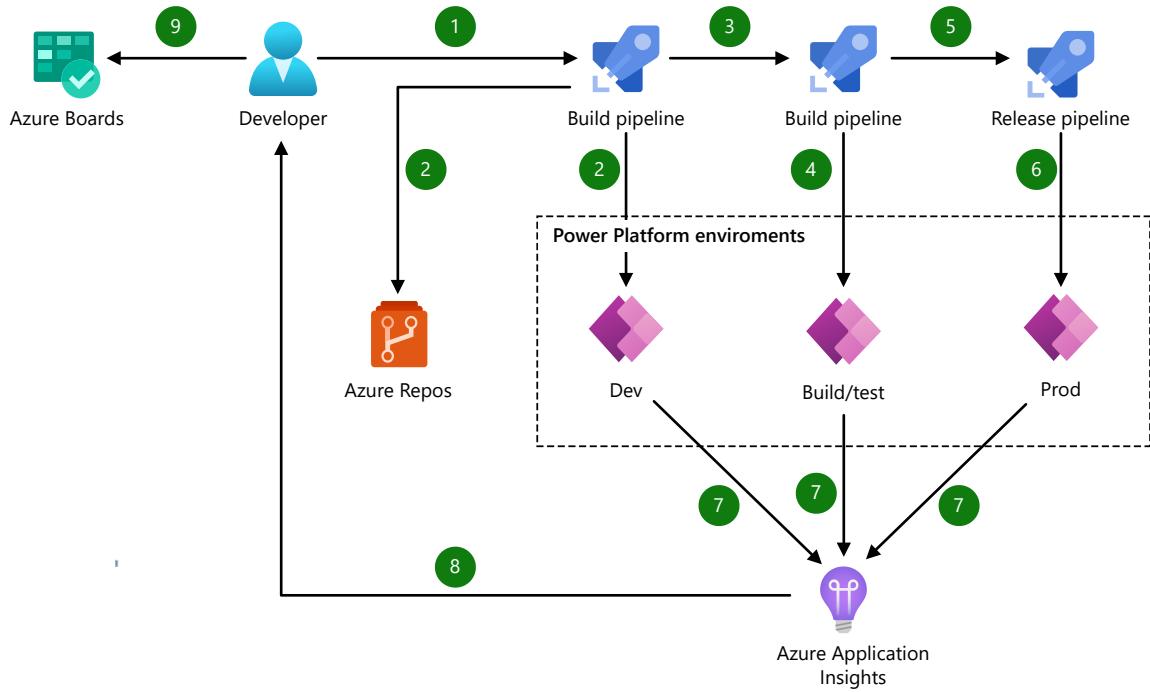
Azure Monitor Azure DevOps Azure App Service Power Apps Azure Repos

💡 Solution ideas

This article is a solution idea. If you'd like us to expand the content with more information, such as potential use cases, alternative services, implementation considerations, or pricing guidance, let us know by providing [GitHub feedback](#).

Learn how to create a CI/CD pipeline to manage your Power Platform Application lifecycle using Azure DevOps.

Architecture



[Download a Visio file](#) of this architecture.

Dataflow

1. The solution is updated, which triggers the build pipeline.
2. Continuous integration exports the solution from the development environment and commits files to Azure Repos.
3. Continuous integration builds a managed solution, runs tests, and creates a build artifact.
4. You deploy to your build/test environment.
5. Continuous deployment runs tests and orchestrates the deployment of the managed solution to the target environments.
6. You deploy to the production environment.
7. Application Insights collects and analyzes health, performance, and usage data.
8. You review the health, performance, and usage information.
9. You update your backlog item(s), as required.

Components

- [Power Apps](#): Microsoft Power Apps is a low-code app-building platform.
- [Azure DevOps](#): Azure DevOps can build, test, and deploy a solution in any language, to any cloud or on-premises.
- [Azure Repos](#): Azure Repos provides cloud-hosted private Git repos.
- [Azure Application Insights](#): Application Insights is a feature of Azure Monitor, which you can use to monitor your live applications.

Scenario details

This architecture enables you to use Azure DevOps, Azure Repos, and Azure Application Insights (via Azure Monitor) to build a CI/CD (continuous integration/continuous deployment) pipeline for Microsoft Power Platform (namely Power Apps).

Potential use cases

- Applications that interact with other Microsoft 365 services.
- Employee onboarding application.
- Image processing tools.
- New user setup scenarios.
- Service request applications.
- Applications with complex entity relationships.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

- [Customize and get pricing estimates ↗](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Sarah Parkes ↗](#) | Cloud Solution Architect

Next steps

- Application lifecycle management with Microsoft Power Platform
- Microsoft Power Platform Build Tools for Azure DevOps
- Microsoft Power Platform Build Tools tasks
- [Azure DevOps ↗](#)

Related resources

Additional Power Apps architectures:

- [Line of business extension](#)
- [Web and mobile front ends](#)

Additional CI/CD architectures:

- [CI/CD for Azure VMs](#)
- [Immutable infrastructure CI/CD using Jenkins and Terraform on Azure virtual architecture](#)
- [Gridwich CI/CD pipeline](#)
- [CI/CD pipeline for container-based workloads](#)

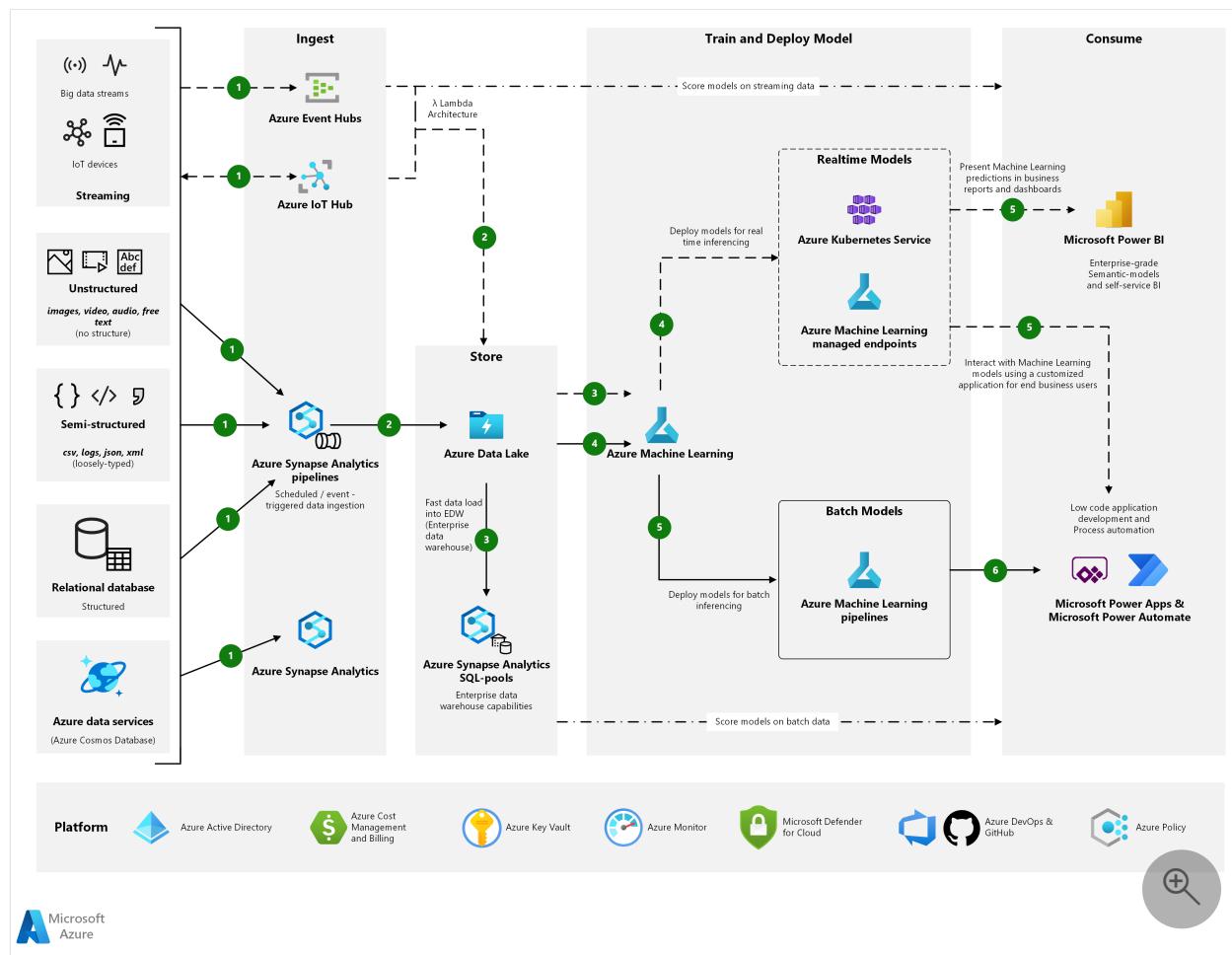
- CI/CD for containers
- Container CI/CD using Jenkins and Kubernetes on Azure Kubernetes Service (AKS)
- Design a CI/CD pipeline using Azure DevOps
- CI/CD for Azure Web Apps
- Java CI/CD using Jenkins and Azure Web Apps
- End-to-end governance in Azure when using CI/CD
- DevSecOps in GitHub

Citizen AI with Power Platform

Azure Machine Learning Microsoft Power Platform Power Apps Power Automate Power BI

The following architecture extends the [analytics end-to-end with Azure Synapse Analytics](#) scenario. It allows for a custom machine learning (ML) model to be trained in Azure Machine Learning and implemented with a custom application built by using Microsoft Power Platform.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

The workflow consists of the following steps:

- Ingest
- Store

- Train and deploy a model
- Consume

Ingest

Use [Azure Synapse Pipelines](#) to pull batch data from various sources, both on-premises and in the cloud. This lambda architecture has two data ingestion flows: streaming and batch. They're described here:

- **Streaming:** In the upper half of the preceding architecture diagram are the streaming data flows (for example, big data streams and IoT devices).
 - You can use [Azure Event Hubs](#) or [Azure IoT Hub](#) to ingest data streams generated by client applications or IoT devices. Event Hubs or IoT Hub ingest and store streaming data, preserving the sequence of events received. Consumers can connect to hub endpoints to retrieve messages for processing.
- **Batch:** In the lower half of the architecture diagram, data is ingested and processed in batches such as:
 - Unstructured data (for example, video, images, audio, and free text)
 - Semi-structured data (for example, JSON, XML, CSV, and logs)
 - Structured data (for example, relational databases and Azure Data Services)

[Azure Synapse Link](#) creates a tight seamless integration between Azure Cosmos DB and Azure Synapse Analytics. [Azure Synapse Pipelines](#) can be triggered based on a predefined schedule or in response to an event. They can also be invoked by calling REST APIs.

Store

Ingested data can land directly in raw format and then be transformed on the [Azure Data Lake](#). Data once curated and transformed to relational structures can be presented for consumption in [Azure Synapse Analytics](#).

Train and deploy a model

[Machine Learning](#) provides an enterprise-grade ML service for building and deploying models faster. It provides users at all skill levels with a low-code designer, automated ML, and a hosted Jupyter notebook environment. Models can be deployed either as real-time endpoints on [Azure Kubernetes Service](#) or as a Machine Learning managed endpoint. For batch inferencing of ML models, you can use [Machine Learning pipelines](#).

Consume

A batch or real-time model published in Machine Learning can generate a REST endpoint that can be consumed in a [custom application built by using the low-code Power Apps platform](#). You can also call a [real-time Machine Learning endpoint from a Power BI report](#) to present predictions in business reports.

Note

Both Machine Learning and Power Platform stack have a range of built-in connectors to help ingest data directly. These connectors might be useful for a one-off minimum viable product (MVP). However, the "Ingest" and "Store" sections of the architecture advise on the role of standardized data pipelines for the sourcing and storage of data from different sources at scale. These patterns are typically implemented and maintained by the enterprise data platform teams.

Components

You can use the following components.

Power Platform services

- [Power Platform](#): A set of tools for analyzing data, building solutions, automating processes, and creating virtual agents. It includes Power Apps, Power Automate, Power BI, and Power Virtual Agents.
- [Power Apps](#): A suite of apps, services, connectors, and data platform. It provides a rapid application development environment to build custom apps for your business needs.
- [Power Automate](#): A service that helps you create automated workflows between your favorite apps and services. Use it to synchronize files, get notifications, collect data, and so on.
- [Power BI](#): A collection of software services, apps, and connectors that work together to turn your unrelated sources of data into coherent, visually immersive, and interactive insights.

Azure services

- [Machine Learning](#): An enterprise-grade ML service for building and deploying models quickly. It provides users at all skill levels with a low-code designer,

automated ML, and a hosted Jupyter notebook environment to support your own preferred IDE of choice.

- [Machine Learning managed endpoints](#): Online endpoints that enable you to deploy your model without having to create and manage the underlying infrastructure.
- [Azure Kubernetes Service](#): ML has varying support across different compute targets. Azure Kubernetes Service is one such target, which is a great fit for enterprise grade real-time model endpoints.
- [Azure Data Lake](#): A Hadoop-compatible file system. It has an integrated hierarchical namespace and the massive scale and economy of Azure Blob Storage.
- [Azure Synapse Analytics](#) : A limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics.
- [Event Hubs](#) and [IoT Hub](#) : Both services ingest data streams generated by client applications or IoT devices. They then ingest and store streaming data, preserving the sequence of events received. Consumers can connect to the hub endpoints to retrieve messages for processing.

Platform services

To improve the quality of your Azure solutions, follow the recommendations and guidelines in the [Azure Well-Architected Framework](#). The framework consists of five pillars of architectural excellence:

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

To create a design that respects these recommendations, consider the following services:

- [Microsoft Entra ID](#) : Identity services, single sign-on, and multifactor authentication across Azure workloads.
- [Azure Cost Management and Billing](#) : Financial governance over your Azure workloads.
- [Azure Key Vault](#) : Secure credential and certificate management.
- [Azure Monitor](#) : Collection, analysis, and display of telemetry from your Azure resources. Use Monitor to proactively identify problems to maximize performance and reliability.
- [Microsoft Defender for Cloud](#) : Strengthen and monitor the security posture of your Azure workloads.

- [Azure DevOps](#) & [GitHub](#): Implement DevOps practices to enforce automation and compliance of your workload development and deployment pipelines for Azure Synapse Analytics and Machine Learning.
- [Azure Policy](#): Implement organizational standards and governance for resource consistency, regulatory compliance, security, cost, and management.

Alternatives

A machine learning MVP benefits from speed to outcome. In some cases, the needs of a custom model can be met by pretrained [Azure Cognitive Services](#) or [Azure Applied AI Services](#). In other cases, [Power Apps AI Builder](#) might provide a fit for a purpose model.

Scenario details

A general technology trend is the growing popularity of citizen AI roles. Such roles are business practitioners looking to improve business processes through the application of ML and AI technologies. A significant contributor to this trend is the growing maturity and availability of low-code tools to develop ML models.

Because of a well-known high failure rate to such initiatives, the ability to rapidly prototype and validate an AI application in a real-world setting becomes a key enabler to a fail-fast approach. There are two key tools for developing models that modernize processes and drive transformative outcomes:

- **An ML toolkit for all skill levels**
 - Supports no-code to fully coded ML development
 - Has a flexible, low-code GUI
 - Enables users to rapidly source and prep data
 - Enables users to rapidly build and deploy models
 - Has advanced, automated ML capabilities for ML algorithm development
- **A low-code application development toolkit**
 - Enables users to build custom applications and automation workflows
 - Creates workflows so that consumers and business processes can interact with an ML model

[Machine Learning](#) fulfills the role of a low-code GUI for ML development. It has automated ML and deployment to batch or real-time endpoints. [Power Platform](#), which includes [Power Apps](#) and [Power Automate](#), provides the toolkits to rapidly build a custom application and workflow that implements your ML algorithm. Business

users can now build production-grade ML applications to transform legacy business processes.

Potential use cases

These toolkits minimize the time and effort needed to prototype the benefits of an ML model on a business process. You can easily extend a prototype to a production-grade application. The uses for these techniques include:

- **Manufacturing Ops with legacy applications that use outdated deterministic predictions.** Such situations can benefit from the improved accuracy of an ML model. Proving improved accuracy requires both a model and development effort to integrate with legacy systems on-premises.
- **Call Center Ops with legacy applications that don't adjust when data drifts.** Models that automatically retrain might provide a significant uplift in churn prediction or risk profiling accuracy. Validation requires integration with existing customer relationship management and ticket management systems. Integration could be expensive.

Considerations

When you use these services to create a proof of concept or MVP, you're not finished. There's more work to make a production solution. Frameworks such as the [Well-Architected Framework](#) provide reference guidance and best practices to apply to your architecture.

Availability

Most of the components used in this example scenario are managed services that scale automatically. The [availability of the services ↗](#) used in this example varies by region.

Apps based on ML typically require one set of resources for training and another for serving. Resources required for training generally don't need high availability, as live production requests don't directly hit these resources. Resources required for serving requests need high availability.

DevOps

DevOps practices are used to orchestrate the end-to-end approach used in this example. If your organization is new to DevOps, the [DevOps Checklist](#) can help you get started.

The [Machine Learning DevOps Guide](#) presents best practices and learnings on adopting ML operations (MLOps) in the enterprise with Machine Learning.

DevOps automation can be applied to the Power Platform solution provided in this example. For more information about Power Platform DevOps, see [Power Platform Build Tools for Azure DevOps: Power Platform](#).

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost-optimization pillar](#).

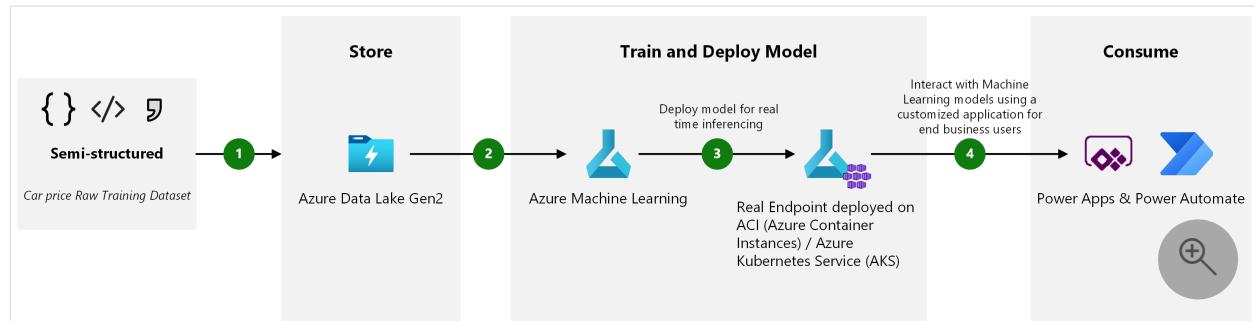
Azure pricing: First-party infrastructure as a service (IaaS) and platform as a service (PaaS) services on Azure use a consumption-based pricing model. They don't require a license or subscription fee. In general, use the [Azure pricing calculator](#) to estimate costs. For other considerations, see [Cost optimization](#) in the Well-Architected Framework.

Power Platform pricing: [Power Apps](#), [Power Automate](#) and [Power BI](#) are SaaS applications and have their own pricing models, including per app plan and per user.

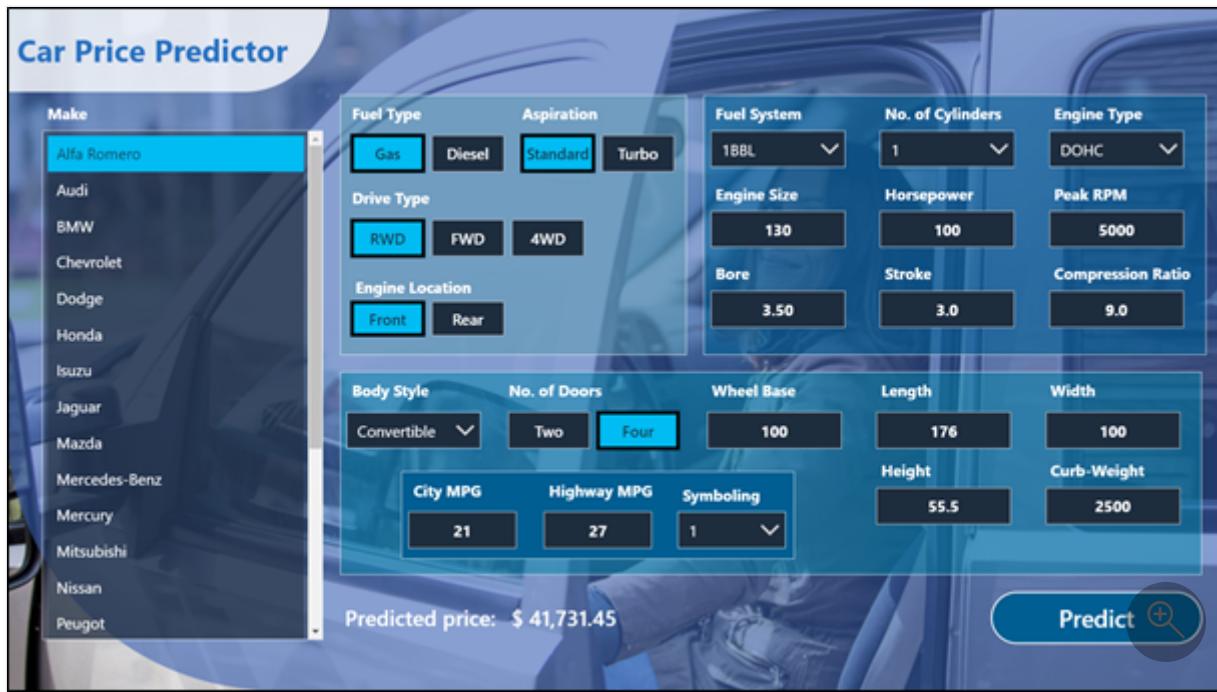
Deploy this scenario

Consider this business scenario. A field agent uses an app that estimates a car's market price. You can use Machine Learning to quickly prototype an ML model of this app. You use a low-code designer and ML features to create the model, and then deploy it as a real-time REST endpoint.

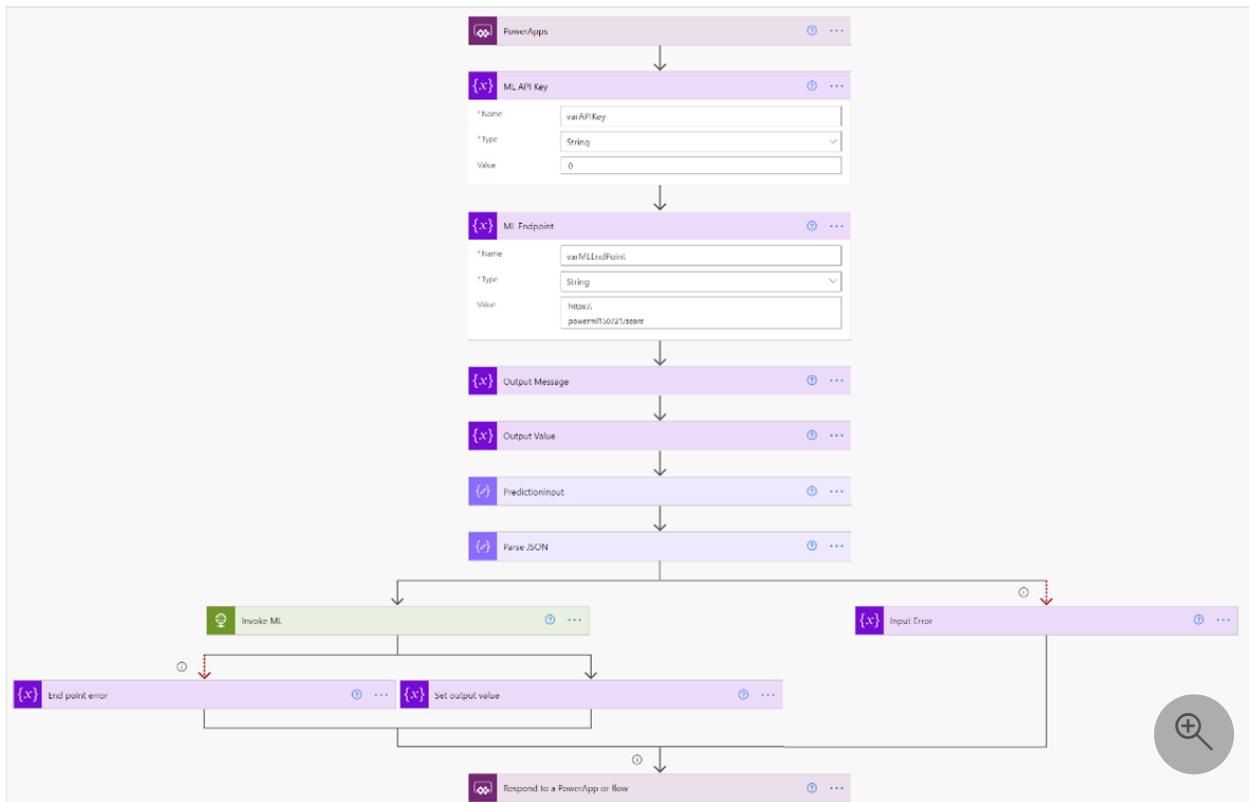
The model might prove the concept, but a user has no easy way to consume a model implemented as a REST API. Power Platform can help close this last mile, as represented here.



Here's a user interface for the app, created in Power Apps by using the low-code interface that Power Apps provides.



You can use Power Automate to build a low-code workflow to parse the user's input, pass it to the Machine Learning endpoint, and retrieve the prediction. You can also use [Power BI to interact with the Machine Learning model](#) and create custom business reports and dashboards.



To deploy this end-to-end example, follow step by step instructions in [Car Price Predictor - Azure ML + Power App Solution](#).

Extended scenarios

Consider the following scenarios.

Deploy to Teams

The sample app provided in the preceding example can also be deployed to Microsoft Teams. Teams provides a great distribution channel for your apps and provides your users with a collaborative app experience. For more information about how to deploy an app to Teams by using Power Apps, see [Publish your app by using Power Apps in Teams: Power Apps](#).

Consume the API from multiple apps and automations

In this example, we configure a Power Automate cloud flow to consume the REST endpoint as an HTTP action. We can instead set up a custom connector for the REST endpoint and consume it directly from Power Apps or from Power Automate. This approach is useful when we want multiple apps to consume the same endpoint. It also provides governance by using the connector data loss prevention (DLP) policy in the Power Platform admin center. To create a custom connector, see [Use a custom connector from a Power Apps app](#). For more information on the Power Platform connector DLP, see [Data loss prevention policies: Power Platform](#).

Contributors

This article is maintained by Microsoft. It was originally written by:

- [Vyas Dev Venugopalan](#) | Sr. Specialist - Azure Data & AI

Next steps

- [How Machine Learning works: Architecture and concepts](#)
- [Build intelligent applications infused with world-class AI](#)

Related resources

- [Analytics end-to-end with Azure Synapse Analytics](#)
- [End-to-end manufacturing using computer vision on the edge](#)
- [Artificial intelligence \(AI\)](#)
- [Compare the ML products and technologies from Microsoft](#)
- [Machine learning operations \(MLOps\) framework to upscale ML lifecycle with Machine Learning](#)

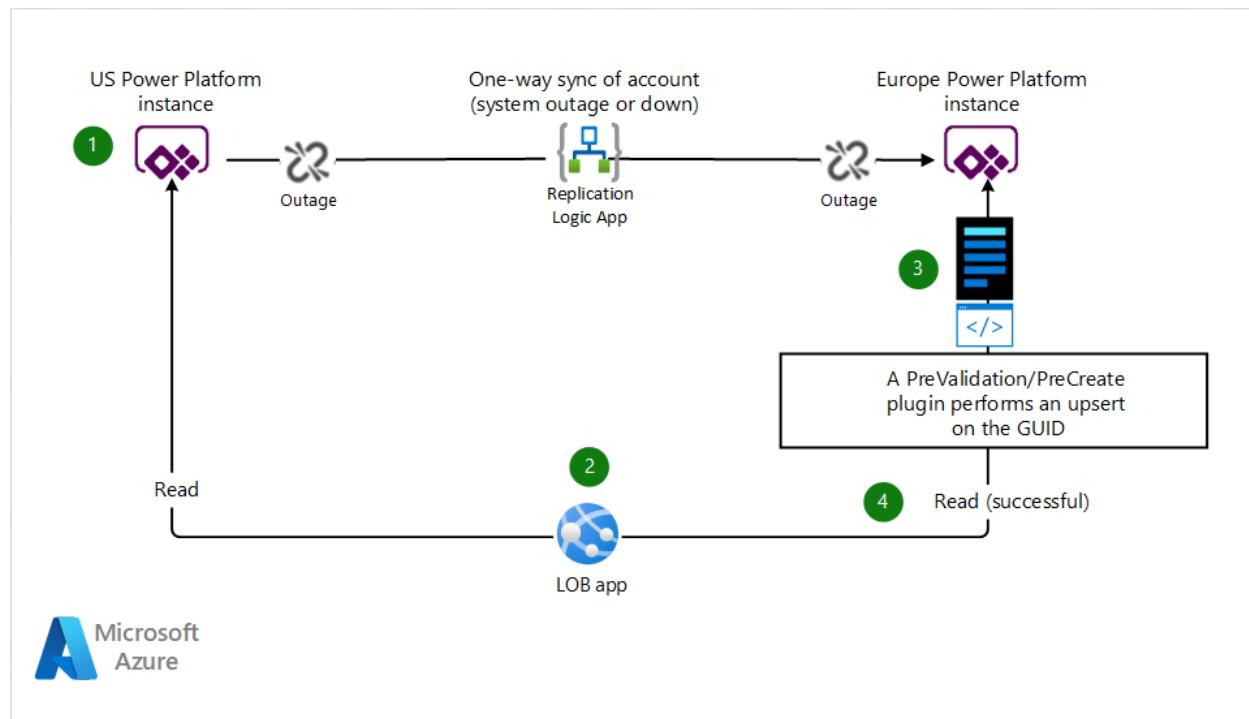
Eventual consistency between multiple Power Apps instances

Microsoft Power Platform Microsoft Dataverse Azure Logic Apps

This article outlines a scenario in which a hypothetical US-based customer, Contoso, has recently acquired another company based in Europe and is in the process of CRM and ERP systems between the two companies. As part of this integration, they must keep a portion of their Dynamics 365 Dataverse entities in sync until they can be fully integrated. A Conotso proprietary line-of-business (LOB) app consumes data from both systems and must be able to accept requests when the data is awaiting synchronization or when it's missing. The following guide shows how to account for eventual consistency between Power Platform instances.

Architecture

Plugin/flow to always upsert based on the GUID or alternate key



Download a [Visio file](#) of this architecture.

Workflow

This solution can be built with several [plugin](#) steps, within the plugin lifecycle. When the entity that you're creating is mandatory, use the [PreValidation step](#). **PreValidation** happens before any database transactions are started. It's the preferred option, if the field is mandatory. However, in some scenarios, a [PreCreate](#) plugin step will suffice.

1. The **US Instance** attempts to synchronize a new account to the **Europe Instance** via a Logic App. The **Europe Instance** is unreachable, due to downtime or upgrade.
2. The Contoso LOB app reads the main account entities from the **US Instance**. It intends to submit an API call that references an account entity that wasn't replicated to the **Europe Instance**. As it stands, the API call would fail because the record doesn't exist, due to the sync not working.
3. However, a **PreValidation/PreCreate** plugin first performs an *upsert* based on the provided entity GUID and provided reference data. If it exists already, then nothing is changed. If it doesn't exist, a new account is created, with most of the fields blank.
4. The API call succeeds because the account with the given ID exists in the system. The plugin intercepted the operation and handled the missing record gracefully. The report from the LOB application is generated successfully.

Note

Microsoft recommends introducing a circuit breaker pattern in your custom code to back off and retry as part of this solution to handle platform outages when referencing either instance. For more information about using a circuit breaker, see [Circuit Breaker Pattern](#).

Alternatives

The scenario described above uses a custom Logic App as the replication method. However, there are multiple ways to replicate data between Dataverse instances, which include, but aren't limited to:

- Logic Apps
- Function apps in Azure Functions
- Azure Data Factory
- Azure Synapse Analytics
- Power Automate

Scenario details

Organizations occasionally find the need to keep two or more Power Platform instances in sync, more specifically, usually a subset of Dataverse entities. This requirement can happen when an organization has intentionally added new instances for geographic isolation but needs a common data set across all geos. Or it can happen when two organizations merge before Power Platform consolidation is complete.

When the syncing process works as designed, line of business applications that consume from both instances don't have issues. However, syncing mechanisms are never error proof, outages or unexpected issues will likely arise. In that case, your line of business application that consumes data from both instances must be built to handle incomplete data.

In order for Contoso's new European subsidiary to be integrated into Contoso's business structure, they must synchronize accounts and contacts from one instance of Power Platform to another. In this scenario, the US instance of Power Platform syncs a daily batch of reference data via a custom Logic App to the European instance. A proprietary Contoso LOB app generates reporting on problem tickets that users have created. To complete this task, the LOB app reads user data from both Dataverse instances to pull the relevant data, the primary reference keys from the US instance and the ticket data from the Europe instance. If the synchronization process hasn't been completed due to downtime, maintenance, or another communications issue, the request will produce an error due to entities missing from the Europe instance.

Potential use cases

This pattern can be useful in the following situations:

- The system that sends reference data is down.
- The synchronization of data takes a long time or the process is delayed.
- Consuming systems have no logic on the creation of the entity being created.

When to use this approach

Use this approach in the following scenarios:

- You want to guarantee a record with a given key exists, and you don't care that the record isn't fully hydrated.
- You must accept creation, even if the data is still not synchronized.

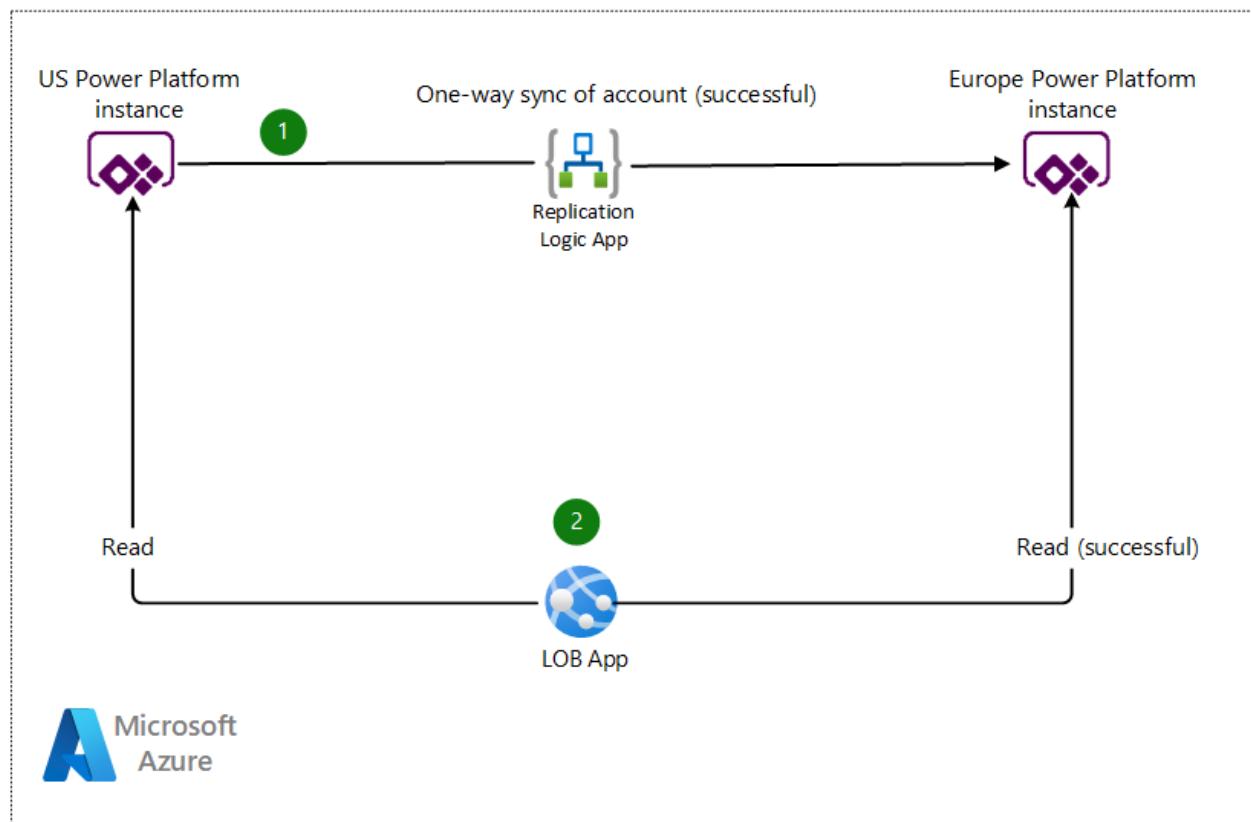
This pattern may not be suitable in the following scenario:

- Logic is applied when the record is created. Because the data won't be hydrated, it's not safe to rely on certain properties being available.

Examples

The following examples show the potential journeys and the result of synchronization delays.

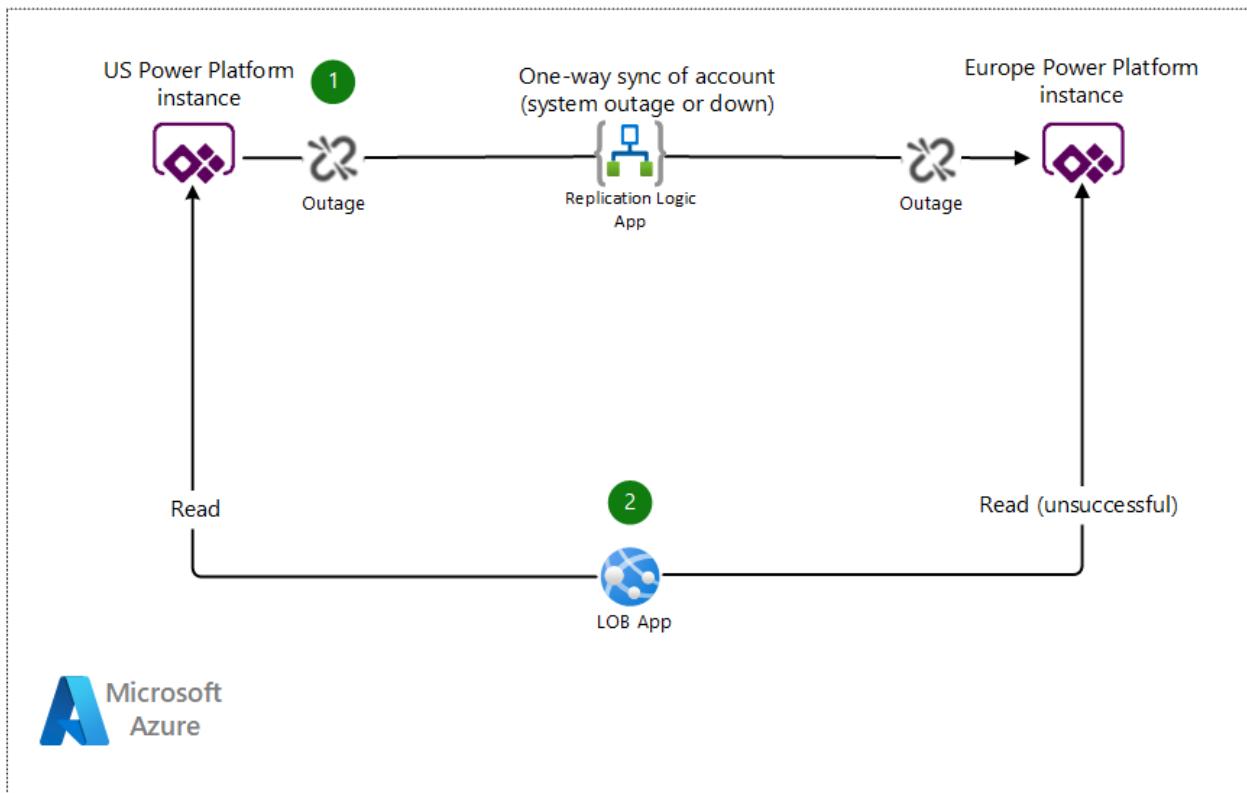
Example 1 - Successful path with no outage or transient errors



Download a [Visio file](#) of this architecture.

1. The **US Instance** synchronizes a new account to the **Europe Instance** via a Logic App. All are working because no transient faults or outages have occurred.
2. The Contoso LOB app reads the main account entities from the **US Instance** and intends to submit an API call that references an account entity that was replicated to the **Europe Instance**. It works because everything was up, and no outages or transient faults occurred. The report from the LOB application is generated successfully.

Example 2 - Unsuccessful path where sync is down or delayed



[Download a Visio file](#) of this architecture.

1. The **US Instance** attempts to synchronize a new account to the **Europe Instance** via a Logic App. The **Europe Instance** is unreachable, due to downtime, maintenance or another communications issue.
2. The Contoso LOB app reads the main account entities from the **US Instance** and intends to submit an API call that references an account entity that wasn't replicated to the **Europe Instance**. The API call fails because the account with the given identifier wasn't created in the **Europe Instance** and the report isn't generated.

Considerations

Consider the impact of any business logic on an entity that isn't hydrated yet. Consider a scenario where the entity isn't fully hydrated and synchronized yet. Some of the properties will be null, so you need to ensure that any decisions on the data are factored in when using this approach.

Next steps

- [Power Platform](#)
- [What is Power Apps?](#)
- [What is Azure Logic Apps?](#)
- [Get started with Power Automate](#)

Related resources

Related architectures:

- CI/CD for Microsoft Power Platform
- Citizen AI with the Power Platform
- Power Automate deployment at scale

Guidance for Web development:

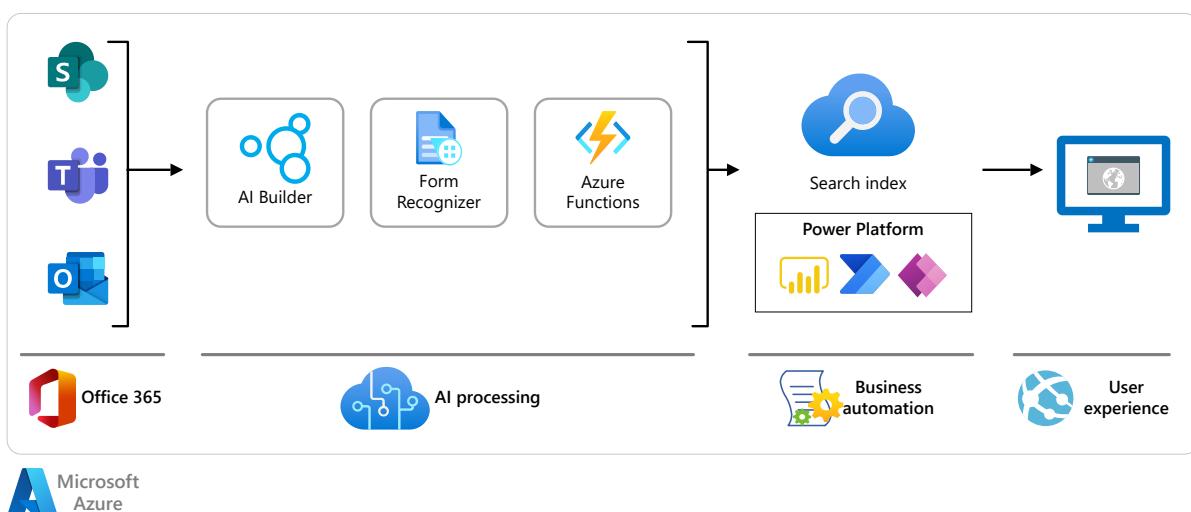
- Ten design principles for Azure applications
- Design and implementation patterns
- App Service deployment best practices
- Microsoft Azure Well-Architected Framework

Extract text from objects using Power Automate and AI Builder

AI Builder Azure AI Document Intelligence Power Automate Microsoft Power Platform Azure Functions

This article presents a solution for extracting text from images so it can be indexed and retrieved in SharePoint. By using AI Builder and Azure Form Recognizer, you can configure a Power Automate workflow to use a trained model to extract text from an image. Once you've configured a workflow, you can quickly search documents for meaningful text that's embedded in shapes and objects.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

1. An object detection model is trained in AI Builder to recognize objects that a user specifies.
2. A new document enters a SharePoint document library, OneDrive, or Teams.
3. The document's arrival triggers a Power Automate event. That event:
 - a. Runs the AI Builder model. AI Builder returns a JSON file that contains the pixel coordinates of any specified objects.
 - b. Sends the document to Form Recognizer for a full optical character recognition (OCR) scan. Form Recognizer returns a JSON file that contains scanned-in text and pixel coordinates of the text.

- c. Runs a function in Azure Functions. The function analyzes the pixel coordinates in the AI Builder and Form Recognizer output files. If detected objects intersect with scanned-in text, the function returns the matched data in a JSON file.
 - d. Enters the metadata, or the text from detected objects, into a document library.
4. The metadata is captured in a SharePoint search index.
5. Users search for the metadata by using PnP Modern Search web parts.

Components

- [AI Builder](#) is a Power Platform capability. Use AI Builder to train models to recognize objects in images. AI Builder also offers prebuilt models for object detection.
- [Form Recognizer](#) uses machine-learning models to extract and analyze form fields, text, and tables from your documents.
- [Power Automate](#) is a part of Power Platform's no-code or low-code intuitive solutions. Power Automate is an online workflow service that automates actions across apps and services.
- [Azure Functions](#) is an event-driven serverless compute platform. Azure Functions runs on demand and at scale in the cloud.
- [PnP Modern Search](#) solution is a set of SharePoint Online modern web parts. By using these tools, you can create highly flexible and personalized search-based experiences.

Alternatives

- Azure Cognitive Services can do a full OCR scan of documents, with the resulting metadata stored in SharePoint.
- SharePoint can run OCR scans on documents and add content output to the index for retrieval. Use search techniques to target key information in documents.
- If you want to process a high rate of documents, consider using Azure Logic Apps to configure the components. Azure Logic Apps prevents you from hitting consumption limits in your tenant, and is cost-effective. For more information, see [Azure Logic Apps](#).

Scenario details

Schematic and industrial diagrams often have objects that contain text. Manually scanning documents for relevant text can be laborious and time consuming.

Potential use cases

Use cases include:

- Complicated engineering schematic diagrams that contain various types of objects. By using this solution, you can quickly search for specific components on a diagram. Having access to embedded text in objects is helpful for investigations, exposing shortages, or looking for recall and failure notices.
- Industrial diagrams that show the components in a manufacturing assembly. This solution promptly identifies pumps, valves, automated switches, and other components. Identifying components helps with preventative maintenance, isolating hazardous components, and increasing the visibility of risk management in your organization.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Consider these points when you analyze and process documents:

- AI Builder can only capture square coordinates when using a trained model. Objects with text outside their boundaries, like triangles and circles, could potentially add unwanted and unnecessary information.
- The metadata that's output from Azure Functions can contain extra characters if there's text outside the object's boundaries.
- The AI Builder creation process can tag more than one object. The resulting JSON file from Azure Functions contains all object types and text. The application consumes the metadata and needs to parse through and process the results.

Availability

Azure replicates data to ensure durability and high availability. Data redundancy protects you from planned and unplanned events, including transient hardware failures, network or power outages, and natural disasters. Choose to replicate your data within the same data center, across zonal data centers within the same region, or across geographically separated regions.

Scalability

Azure Functions is highly scalable. This platform offers multiple plans that automatically scale on demand when events are triggered. For more information, see [Event-driven](#)

scaling.

Azure Functions has a limit of 200 instances. If you need to scale beyond this limit, add multiple regions or app plans.

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Overview of the security pillar](#).

Use standard security practices for the components that you use, and for the SharePoint document library that you store the metadata in.

Form Recognizer is designed with compliance, privacy, and security in mind. It authenticates access by using an API key, encrypts data during transit and storage, and returns results by using the API key. For more information, see [Data, privacy, and security for Form Recognizer](#).

AI Builder relies on environment security and Dataverse security roles and privileges to grant access to AI features in Power Apps. Privileges are set by default in Dataverse. System administrators can use the default built-in security roles without further actions. For more information, see [Security overview](#).

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

- For Power Automate, make sure the licenses that you've purchased and assigned are adequate for the volume of documents that you process. Include an HTTP premium connector to call Form Recognizer and Azure Functions.
- Purchase AI Builder credits based on the expected model usage.
- To estimate the cost of Azure products and configurations, use the [Azure pricing calculator](#).

Deploy this scenario

For more information on deploying this scenario, see the [Power Automate Community Blog](#) and the [Extract Text From Objects GitHub repo](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Steve Pucelik](#) | Sr. Specialist

Next steps

- Understand the types of documents that would be well suited for this solution. Typical documents include schematic diagrams, manufacturing control processes, and diagrams that contain many shapes that need to be isolated. For more information, see [Form Recognizer models](#).
- Become familiar with the capabilities that AI Builder offers. For more information, see [AI Builder in Power Automate overview](#).
- Define an information architecture that can receive and process your metadata. For more information, see [Cognitive Search skill set](#).
- For information on how the solution works and whether it's suitable for your use cases, see [Extract text from objects](#).

Related resources

- [Knowledge mining for content research](#)
- [Vision classifier model with Azure Custom Vision Cognitive Service](#)

Optimize inventory and forecast demand with Power Platform and Azure

Azure Data Factory

Azure Machine Learning

Azure Data Lake Storage

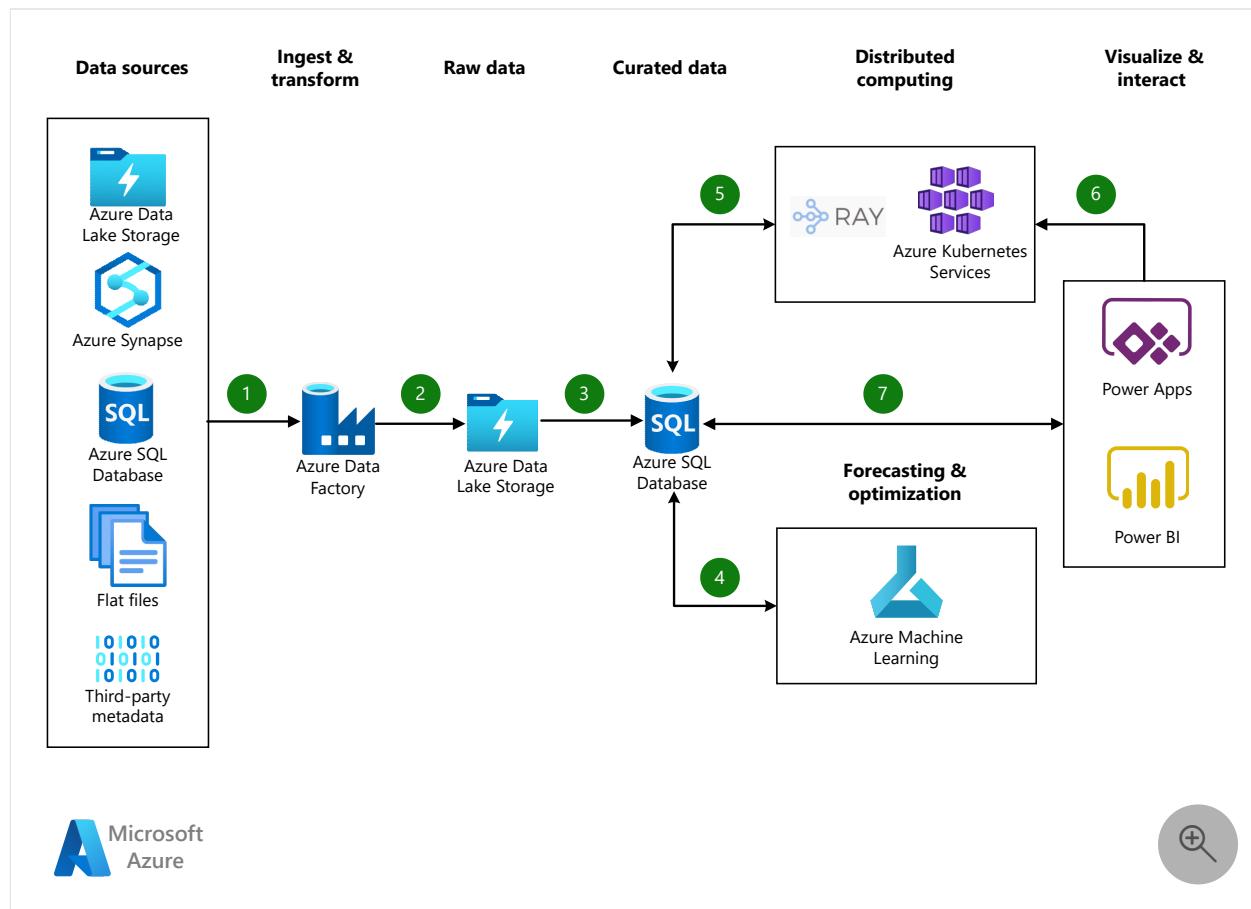
Azure Synapse Analytics

Microsoft Power Platform

This article showcases a practical, scalable, and manageable solution for implementing inventory optimization architectures for the retail industry. The solution uses the latest advancements in forecasting, optimization, and parallel computing.

Ray® or Ray.io® is either a registered trademark or trademarks of the Anyscale, Inc. in the United States and/or other countries. No endorsement by Anyscale, Inc. is implied by the use of these marks.

Architecture



Download a [Visio file](#) of this architecture.

Dataflow

1. Azure Data Factory ingests related data into Azure Data Lake Storage. The sources of this data can be enterprise resource planning (ERP) systems, SAP, and Azure SQL. Additional sources might include weather and economic data.
2. Data Lake Storage stores raw data for further processing.
3. Mapping data flows in Azure Data Factory produces curated data and stores it in a relational format in an Azure SQL database. Additionally, in this use case, the SQL database stores intermediate results, other run information, and simulation metrics. Alternatively, Azure Machine Learning can read the data directly from the Data Lake service.
4. Use Azure Machine Learning to train the model by using data in Azure SQL Database, and deploy the model and service to Kubernetes.
5. Install the Ray framework on the same Kubernetes cluster to parallelize the execution of the scoring script during inferencing. Each execution runs the demand-forecasting module for specified locations and products over a given forecast period. The forecasting results are read by the optimization module, which calculates the optimal inventory levels. Finally, the results are stored in Azure SQL Database.
6. Power Apps hosts a user interface for business users and analysts to collect parametric information, such as service level, product, and location. Users also use Power Apps to submit the collected parameters and to launch executions of the deployed machine learning module that is hosted in Kubernetes clusters.
7. Power BI ingests data from Azure SQL Database and allows users to analyze results and perform sensitive analysis. All Power BI dashboards are integrated into Power Apps to have a unified UI for calling the API, reading results, and performing downstream analysis.

Components

- [Data Lake Storage](#) is a scalable and secure data lake for high-performance analytics workloads. By using Data Lake Storage, you can manage petabytes of data with high throughput. Data Lake Storage can accommodate multiple, heterogeneous sources and data coming in structured, semi-structured, or unstructured formats.

- [Azure Data Factory](#) is a scalable and serverless service that provides a data-integration and transformation layer that works with various data stores.
- [Power BI](#) is a collection of software services, apps, and connectors that work together to turn your unrelated sources of data into coherent, visually immersive, and interactive insights.
- [Azure Machine Learning](#) is a cloud service for accelerating and managing the lifecycle of a machine learning project. Machine learning professionals, data scientists, and engineers can use Azure Machine Learning in their day-to-day workflows, to train and deploy models, and to manage Machine Learning Operations (MLOps).
- [Azure SQL Database](#): Azure SQL Database is a fully managed platform as a service (PaaS) that handles most of the database management functions, such as upgrading, patching, backups, and monitoring, without user involvement.
- [Azure Kubernetes Service \(AKS\)](#): AKS simplifies deploying a managed Kubernetes cluster in Azure by offloading the operational overhead to Azure. As a hosted Kubernetes service, Azure handles critical tasks like health monitoring and maintenance.
- [Azure Synapse Analytics](#): Azure Synapse Analytics is an enterprise analytics service that accelerates time to insight across data warehouses and big data systems. Azure Synapse Analytics brings together the following technologies:
 - The best of the SQL technologies used in enterprise data warehousing
 - Spark technologies used for big data
 - Data Explorer for log and time-series analytics
 - Pipelines for data integration and ETL/ELT
 - Deep integration with other Azure services, such as Power BI, Azure Cosmos DB, and Machine Learning

Alternatives

In this solution, Machine Learning performs forecasting and inventory management analytics. However, you can use [Azure Databricks](#) or Azure Synapse Analytics to perform the same type of analytics when the amount of data is large. For more information about using Azure Synapse Analytics, see [Apache Spark in Azure Synapse Analytics](#).

As an alternative to [mapping data flows](#) in Azure Data Factory to curate and perform ETL on data in [Azure Data Lake](#), you can use Azure Databricks for a code-first approach.

Depending on your specific use case and your choice of analytics platform for end users, you can use other relational or storage services, such as Azure Synapse Analytics or [Data Lake Storage](#), instead of storing your data in Azure SQL. For example, if the data has accumulated for a long period of time and there's a need to run analytics queries against this data, Azure Synapse analytics is a good option as part of the architecture.

Instead of running the Ray framework on Kubernetes, you can use the Ray framework on a compute instance in Azure Machine Learning to perform inferencing. If you incorporate the Ray framework on Azure Machine Learning, you might find [ray-on-ml](#), a package on GitHub, helpful.

You could use [Web Apps](#) instead of, or along with, Power Apps to create the user interface for access to the Power BI embedded reports.

Scenario details

Retail competition between companies is intense, and retailers are required to bring computational intelligence and smart insights into their commercial, marketing, and manufactory processes. While increasing and retaining customers is paramount for the success of a company, retailers must also align their inventory levels to customer demand to maximize operational efficiency and reduce waste and costs.

Inventory optimization is the process of providing the right inventory, in the right quantities, at the right locations to meet the customer demand. The objective of optimization is to lessen the carrying, storing, and maintenance costs, while meeting the needs of customers and maintaining a high level of customer satisfaction. *Demand forecasting* is the process of making predictions about customer demand by using various historical and third-party data. Demand forecasting helps business leaders to make informed decisions and is one of the most widely used techniques of inventory optimization.

To better work with a large volume of supply chain data, this solution offers the following guidelines for transforming and scaling existing on-premises solutions:

- Provide a customer interface by using Power BI and PowerApps from which users can launch simulations and determine the parameters of the simulation and data, access results, and so on.
- Process, validate, and analyze data by using Azure Machine Learning.
- Generate probabilistic forecasts of inventory supply levels by using advanced machine learning methods, such as DeepAR.

- Run parallel simulations for generating and forecasting inventory by using Azure Kubernetes and Ray.

Potential use cases

This solution is designed for the retail industry, but it also applies to the manufacturing industry and to the following scenarios:

- Analyze product information across locations to assess demand levels and decrease inventory costs.
- Analyze stock variability and sales by using historical demand data to forecast demand in future periods, across customers, and by location and sales channel. For example, a shipping and delivery company needs to predict the quantities of products at different locations, and an insurer might want to know the number of products that will be returned because of failures.
- Identify the ideal amount of inventory to have in stock.
- Predict how seasonal changes or other events might affect sales and restocking options.
- Forecast the prices of commodities across locations and sales channels by using historical transaction data in a retail context.

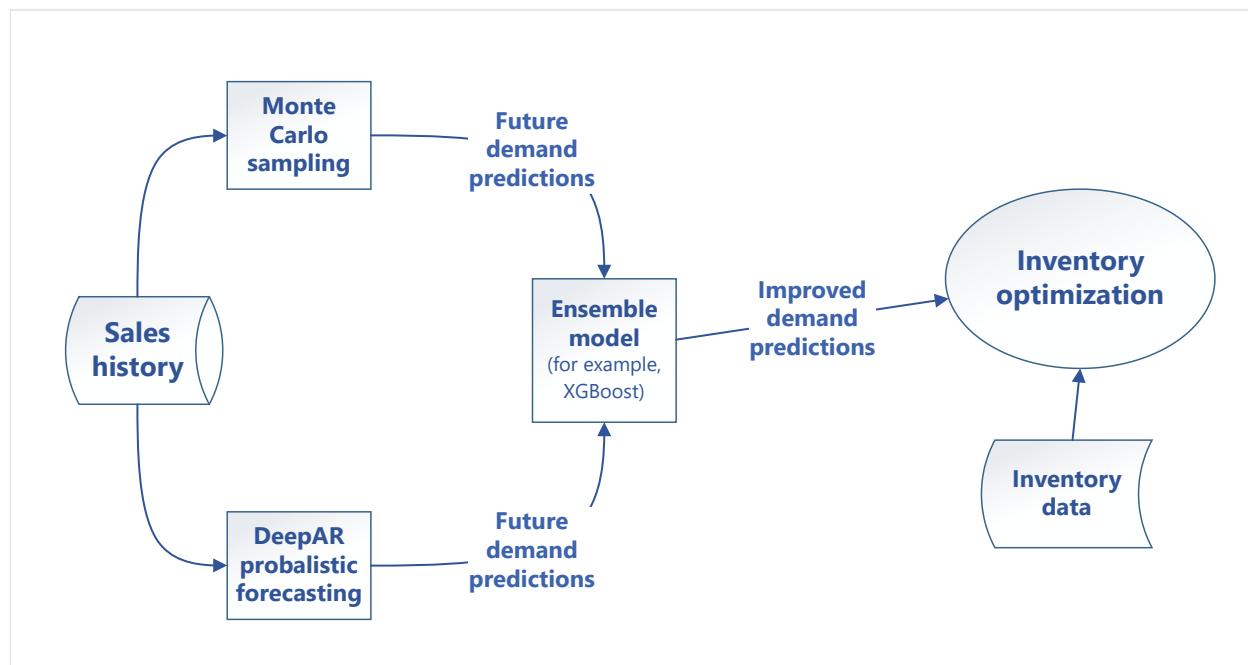
Companies can have a wide variety of inventory types, and specific types might be present only in specific locations or available from a subset of factories. Companies must also meet service level agreements and other relevant metrics. So, forecasts must account for the time at which a specific unit is available at a specific location, in addition to forecasting demand, service level agreements, and other relevant metrics. Successful inventory management requires accurate simulations for forecasting demand, utilization of distributed computing resources, and methodologies that can predict for multiple time granularities, product types, and locations.

Often, the data that's required to optimize inventory is sparse and not centrally located, which makes aggregating and analyzing it difficult. Most companies rely on commercial software. However, such systems hit scalability limits due to the ever-increasing amount of data and the complexity of data storage systems.

Methods of forecasting demand include on-point predictions, probabilistic Monte Carlo simulations, time-series analysis, and data science methodologies. Some of these methods can take historical sales and seasonality effects into account, but more complex parameters require sophisticated methodologies for high-quality forecasts.

Forecasting with Deep Learning

The implemented reference solution uses advanced machine learning and deep learning methods for time-series forecasting. Specifically, this solution uses multivariate probabilistic forecasting to account for uncertainties that are common in supply chains. By using an ensemble modeling approach that blends Deep AutoRegression RNN (DeepAR) or Transformer models that use the classical Monte Carlo Sampling method, the reference solution achieved 99.9% mean square error (MSE) improvement over a customer's initial approach for high-volume/high-impact business products. For ensembling, this solution explored XGBClassifier and XGBRegressor model architectures.



Considerations

Consider following the Microsoft Azure Well-Architected Framework when deploying the solution to production. The Well-Architected Framework provides technical guidance across five pillars of workload: reliability, security, cost optimization, operational excellence, and performance efficiency. For more information, see [Azure Well-Architected](#).

Follow MLOps guidelines to standardize and manage end to end a scalable Machine Learning lifecycle across multiple workspaces. Before going into production, ensure that the implemented solution supports ongoing inference with retraining cycles and automated redeployments of models. For more information about implementing MLOps in Azure, see [Machine learning DevOps guide](#) and [Azure MLOps \(v2\) solution accelerator](#), a project on GitHub.

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Overview of the security pillar](#).

Consider using Azure Databricks Premium for more security features. For more information, see [Azure Databricks Pricing ↗](#).

Follow the best practices for Databricks security and data governance. For more information, see [Secure cluster connectivity \(No Public IP / NPIP\)](#).

Consider implementing the following additional security features in this architecture:

- [Store credentials in Azure Key Vault](#)
- [Deploy dedicated Azure services into virtual networks](#)

Cost optimization

To estimate the cost of implementing this solution, use the [Azure Pricing Calculator ↗](#) for the services mentioned in this article. You might also find [Overview of the cost optimization pillar](#) to be helpful in planning your strategy to efficiently scale out your architecture and implementation.

Power BI comes with different licensing offerings. For more information, see [Power BI pricing ↗](#).

Depending on the volume of data and complexity of your geospatial analysis, you might need to scale your Databricks cluster configurations that would affect your cost. For best practices on cluster configuration, see the Databricks [cluster sizing](#) examples.

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. For more information, see [Performance efficiency pillar overview](#).

If the amount of input data is large, consider using [Ray Dataset ↗](#) along with Ray framework. Ray datasets provide distributed data transformations on various file formats and are easily integrated with other Ray libraries and applications.

If you use mapping data flows in Azure Data Factory for ETL, follow [the performance and tuning guide](#) to optimize your data pipeline and ensure that your dataflows meet your performance benchmarks.

Often, for optimization and Operations Research problems, compute-intensive calculations run after the inferencing is invoked. If you use the Ray framework for distributed computing, as suggested in this article, make sure to utilize [Ray Dashboard](#) to monitor the execution metrics and increase the node counts in Kubernetes cluster.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Giulia Gallo](#) | Senior Cloud Solution Architect

Other contributors:

- [Chu Lahlou](#) | Senior Cloud Solution Architect
- [Gary Moore](#) | Programmer/Writer
- [Arash Mosharraf](#) | Senior Cloud Solution Architect

Next steps

- [Copy and ingest data using Azure Data Factory](#)
- [Deploy machine learning models to Azure](#)
- [Install Ray on Kubernetes cluster](#)
- [Ray.io framework documentation](#)
- [Ray installation on Kubernetes](#)
- [Ray on Databricks](#)
- [Security baseline for Azure Machine Learning service](#)

Related resources

- [Demand forecasting for shipping and distribution](#)
- [Energy supply optimization](#)
- [Enterprise business intelligence](#)
- [Forecast energy and power demand with machine learning](#)
- [Interactive price analytics using transaction history data](#)
- [MLOps for Python models using Azure Machine Learning](#)

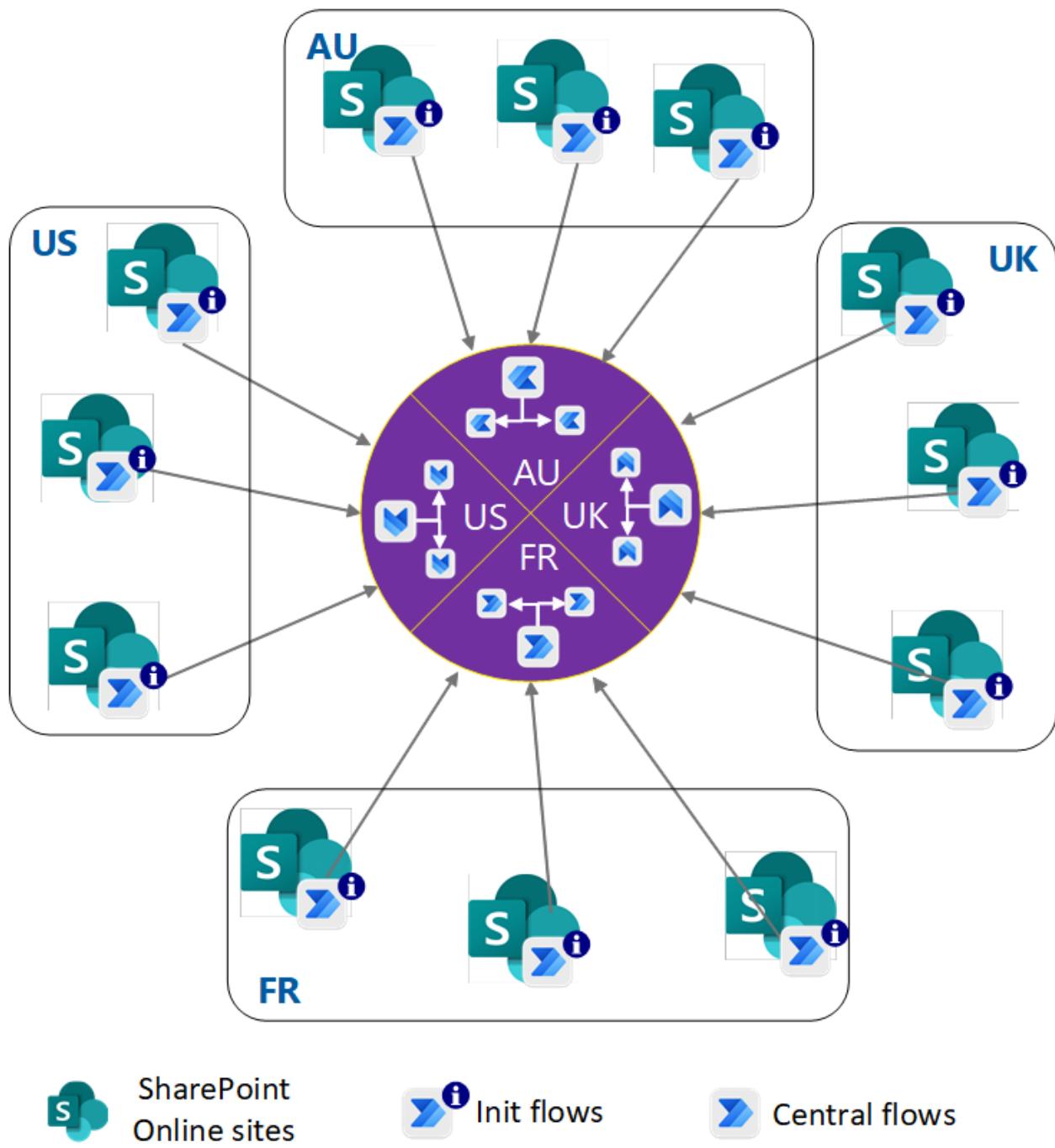
Power Automate deployment at scale

Azure Logic Apps Microsoft 365 SharePoint Power Automate Microsoft Power Platform

This architecture is for Power Automate workflows that replace SharePoint 2010 workflows, and for new SharePoint Online sites. With this solution, you can:

- Carefully plan your Power Automate deployment, governance, and operation strategy.
- Meet organizational needs like data residency requirements, data loss prevention (DLP), and flexible and minimal licensing requirements.
- Stay within the scalable thresholds of Power Platform.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

Azure hub-spoke network topology inspires this architecture. Power Platform [Solutions](#) flows can invoke child flows from Solutions. Parent and child flows ease flow management by avoiding flows with hundreds of steps.

This solution provisions a Power Automate *init flow* workflow for each SharePoint site. In the init flow, either a user or an event *initiates* a workflow. The init flow calls a *central flow* workflow, which runs all the actions that meet a business need.

SharePoint Online and Power Platform support many geographic regions. Each region has a set of SharePoint Online sites. This *multi-geo* concept extends to the central flows.

1. The IT team provisions an init flow on each SharePoint Online site. Besides the trigger, the flow contains a single action that uses `Call Child Flow` to call a central flow workflow.
2. The team provisions the central flows in each geographical region, corresponding to the SharePoint Online regions.
3. A user initiates, or an event triggers, an init flow workflow. Depending on the trigger type, the init flow can execute in the user's context or in the `maker` context.
4. The init flow calls a central flow in the appropriate region. For a user context, the init flow can authoritatively pass along the user information to assert the user to the central flow.
5. The central flow can invoke more child flows if needed to keep the flow lightweight.

Components

This scenario uses the following components:

- [Power Automate](#) uses flows to build automated processes.
- [SharePoint](#) Online sites help organizations share and manage content, knowledge, and applications.
- [Power Platform](#) analyzes data, builds solutions, automates processes, and creates virtual agents.
 - [Power Platform environments](#) store, manage, and share an organization's business data, apps, chatbots, and flows.
 - [Power Platform Solutions](#) are the mechanism for implementing [application lifecycle management \(ALM\)](#) in Power Apps and Power Automate.
- [Microsoft Entra ID](#) is a universal platform to manage and secure identities.

Alternatives

- You can use this architecture with [Azure Logic Apps](#) by replacing the central flows with logic apps. There are some triggers that Logic Apps doesn't have, like *SharePoint - For a Selected File*. In that case, the Power Automate init flow can use the trigger, and then call a logic app.

Logic Apps supports the consumption model, where you pay for what you use. A hybrid model using both Power Automate and Logic Apps is also feasible. If you don't want to worry about thresholds, Logic Apps is the recommended solution.

- You can improve the hub-and-spoke model by using a single init flow per region instead of creating one flow per SharePoint Online site. This strategy is possible only if you trigger the flow manually. You can orchestrate the flow to be invoked from any SharePoint Online site in a tenant.

Scenario details

Microsoft Power Automate is part of the no-code or low-code Microsoft Power Platform. Microsoft 365 customers use Power Automate for workflow automation and business process flows.

Potential use cases

Customer-designed Power Automate workflows fall into two categories:

- SharePoint site owners usually create ad-hoc workflows. Site owners take full responsibility for workflow design, deployment, and maintenance.
- IT teams create workflows that they fully own, manage, and support over the workflow lifetime.

This architecture applies to workflows where IT teams fully control the workflow and component life cycles.

Considerations

Here are some advantages of adopting this hub-and-spoke model for your Power Automate deployments:

- Centralized logic is easy to update in one place, and all the flows automatically get the latest updates.
- You can avoid assigning premium licenses to all the init flows for the SharePoint Online sites. Instead, you can assign premium licenses to the limited number of central flows.
- Segregating SharePoint Online sites and flows into their own regions meets data residency requirements.

- Depending on the init flow trigger, the flow can retain the user's context from initiation to central flow completion.
- To meet seasonal or periodic requirements, this model offers flexible central flow license upgrades and downgrades.

DevOps

- Power Platform supports continuous integration and continuous delivery (CI/CD) for its components in Solutions. You can export and import Solutions as packages across Power Platform environments and across tenants.
- It's best to have a pre-production tenant to validate updates before you push updates and components to your production tenant. Since updating the central flows immediately impacts many init flows, it's important to have high-quality analysis and validation. When promoting to the production tenant, make sure to use environment variables for connections, so you can choose the endpoint corresponding to the target tenant.
- Power Platform supports component and workflow ALM with Azure Pipelines or GitHub Actions.

Operations

Use the [Center of Excellence \(CoE\) toolkit for Power Platform](#) to centrally manage flows and monitor them for failures. The CoE toolkit also shows the Power Platform components and dependencies between components. You can design each flow to catch and log failures or notify someone for better supportability.

Security

- Entitlement management for flows you create under Solutions is different from flows outside Solutions. With flows outside Solutions, you can give permissions to a SharePoint site list or library to initiate the flow. Flows in Solutions tie permissions to a Dataverse Environment-based group called *Group Team*, which you can map to a Microsoft Entra group. You can then manage users in the Microsoft Entra group.
- All users except the environment administrator must be given read/execute-only permissions in Production environments, so end users can't create any components.

- You can apply DLP policies at the environment level, which allows more flexibility to meet business requirements.

Cost optimization

You pay no extra costs for this scenario if you meet the following conditions:

- There's no dependency on Power Platform premium connectors.
- The flows can conform to the action executions thresholds of the Microsoft 365 seed license, such as E3 or E5.

Otherwise, you need to purchase premium licenses, per user or per flow plan, for the central flows only. Pricing may vary depending on how many central flows you need in each geographic region. You don't have to assign premium licenses to the init flows, which are higher in number.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Srinivas Varukala](#) | Senior Technical Program Manager

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- [Use Child Flows in Power Automate](#)
- [Define a Power Platform Environment Strategy](#)
- [Microsoft Dataverse Group Teams](#)
- [Use Environment Variables in Power Platform Solutions](#)
- [ALM with Power Platform](#)
- [Power Automate Pricing](#)
- [Power Automate Licensing FAQ](#)

Related resources

- [CI/CD for Azure Power Platform](#)
- [Citizen AI with the Power Platform](#)

- Hub-spoke network topology in Azure

Build an architecture with real-time machine learning inference and low-code web application UI on Azure

Azure Machine Learning

Microsoft Power Platform

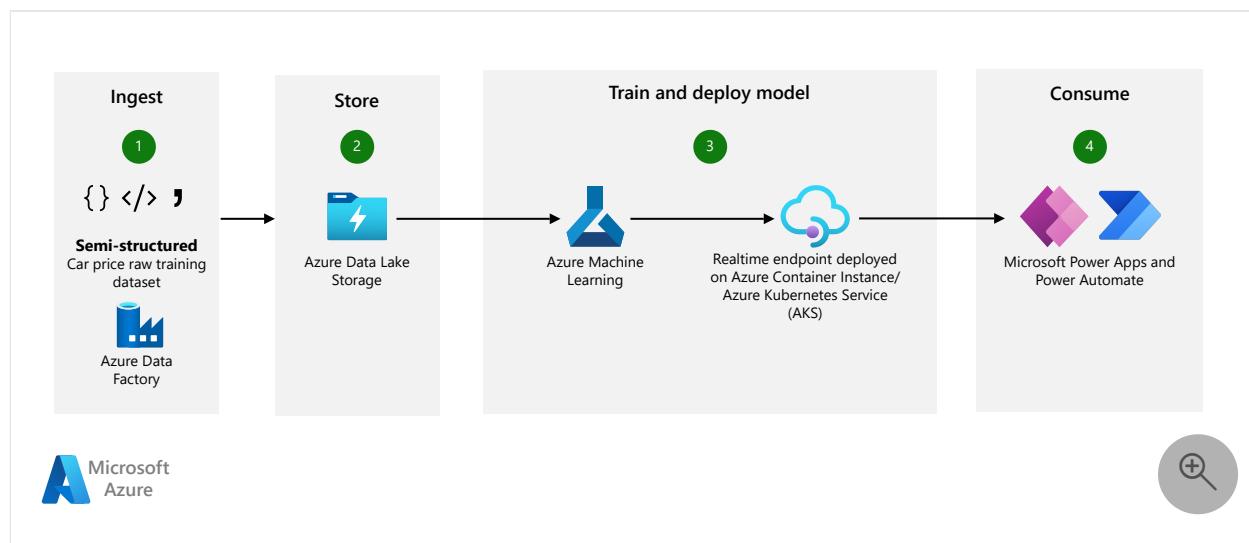
Power Apps

Power Automate

Power BI

This solution expands on [Citizen AI with the Power Platform](#), which provides a high-level example of a low-code, end-to-end *lambda architecture* for real-time and batch data streaming. It covers how to deploy machine learning models for real-time and batch inference. This article also covers how to consume these models by using an end-user application or analyzing results in Power BI.

Architecture



Download a [Visio file](#) of this architecture.

Dataflow

This article guides you through a model-view-presenter (MVP) architecture by using semi-structured data stored in [Azure Data Lake Storage](#). You use this data in [Azure Machine Learning](#) for training a machine learning model. You deploy the model to a real-time endpoint deployed on an [Azure Container Instance](#) or [Azure Kubernetes Service \(AKS\)](#) cluster. Finally, Power Apps consumes the model by using a low-code, custom user app.

1. Ingest: Semi-structured data, like JSON, XML, CSV, and logs, is loaded into Data Lake Storage. You can extend the scope of data ingestion by using [Azure Synapse pipelines](#) to pull batch data from a wide variety of sources. You can extend the scope to more data types—without changing the architecture design—both on-premises and in the cloud. This data includes:

- Unstructured data like video, images, audio, and free text.
- Structured data like relational databases and Azure data services.

2. Store: You can ingest data in a raw format and then transform it in [Data Lake Storage](#).

3. Train and deploy model: Machine Learning provides an enterprise-grade machine learning service to quickly build and deploy models. It provides users at all skill levels with an environment that offers a low-code designer, automated machine learning, and a hosted Jupyter notebook. You can deploy models as real-time [endpoints](#) on AKS or as a [managed online endpoint](#). For batch inferencing of machine learning models, you can use [Machine Learning pipelines](#).

4. Consume: A real-time published model in Machine Learning can generate a REST endpoint that can be consumed in a [custom application that's built by using the low-code Power Apps platform](#). You can also call a [real-time Machine Learning endpoint from a Power BI report](#) to present predictions in business reports.

Note

Both the Machine Learning and the Power Platform stacks have a range of built-in connectors to help ingest data directly. These connectors might be useful for a one-off minimally viable product. However, the **Ingest** and **Store** sections of the architecture promote the role of standardized data pipelines for the sourcing and storage at scale of data from multiple sources. These patterns are typically implemented and maintained by the enterprise data platform teams.

Components

- [Azure Data Lake Storage](#): A Hadoop-compatible file system. It has an integrated hierarchical namespace and the scale and economy of Azure Blob Storage.
- [Azure Machine Learning](#): An enterprise-grade machine learning service used to quickly build and deploy models. It provides users at all skill levels with a low-code designer, automated machine learning, and a hosted Jupyter notebook environment to support your preferred integrated development environment.

- [Azure Kubernetes Service](#) : Machine Learning has varying support across different compute targets. Azure Kubernetes Service is one such target, and it's a great fit for high-scale production deployments. It provides a fast response time and autoscaling of the deployed service.
- [Azure Container Instances](#) : Container Instances is great fit for real-time inference, and it's recommended for development and test purposes only. If you don't need to manage a cluster, use it for low-scale CPU-based workloads that require less than 48 GB of RAM.
- [Microsoft Power Platform](#) : A set of tools for analyzing data, building solutions, automating processes, and creating virtual agents. It includes Power App, Power Automate, Power BI, and Power Virtual Agents.
- [Power Apps](#) : A platform with a suite of apps, services, and connectors. It provides an environment for rapid application development to build custom apps for your business needs.
- [Power Automate](#) : A service that helps you create automated workflows between your favorite apps and services. Use it to synchronize files, get notifications, collect data, and so on.

Alternatives

The solution in this article focuses on an architecture that benefits from speed-to-outcome. In specific use cases, the needs of a custom model can be met by pre-trained models that use [Azure Cognitive Services](#) or [Azure Applied AI Services](#). In others, [Power Apps AI Builder](#) might provide a fit-for-purpose model.

Scenario details

The ability to rapidly prototype and validate an AI application in a real-world setting is important to following a fail-fast approach. The following services can help with this development:

Machine Learning

- Supports no-code to fully coded machine learning development
- Has a flexible, low-code GUI
- Enables users to rapidly source and prep data
- Enables users to rapidly build and deploy models

- Has advanced, automated machine learning capabilities for machine learning algorithm development

Power Apps and Power Automate

- Enables users to build custom applications and automation workflows
- Creates workflows so that consumers and business processes can interact with a machine learning model

End-to-end analytics

This architecture extends [Analytics end-to-end with Azure Synapse](#), an example scenario. With this scenario, a custom machine learning model can train in Machine Learning. Then you can implement the model with a custom application built by using Microsoft Power Platform.

[Machine Learning](#) fulfills the role of a low-code GUI for machine learning development. It has automated machine learning and deploys to batch or real-time endpoints.

[Microsoft Power Platform](#), which includes [Microsoft Power Apps](#) and [Microsoft Power Automate](#), provides the tools to rapidly build a custom application and workflow that implements your machine learning algorithm. Now your end users can build production grade machine learning applications to transform their legacy business processes.

Potential use cases

This example workload is designed to help a buyer or a purchasing agent in the automotive industry estimate a car's market price. A user can use a Power App to submit vehicle details to a model that's trained on market data and receive a price prediction in return.

The applicability of this example workload *isn't limited to a specific industry and can apply to a variety of use cases*. Any use case that uses data stored on a data lake for model training and deployment to a real-time web application can also be used for unstructured or structured data.

- **Customer segmentation:** Identify target markets based on real-time data and indicators. For example, predict the promotion that a shopper might respond to based on purchase data and customer details.
 - Key industries: Banking, insurance, retail, and telecommunications.
- **Churn prevention:** Identify signs of dissatisfaction among customers and identify customers who are at risk for leaving.

- Key industries: Banking, insurance, automotive, and retail
- **Predictive maintenance:** With operational reporting, and by analyzing metrics and real-time data related to the lifecycle maintenance of technical equipment, companies can predict timelines, potential maintenance events, and upcoming expenditure requirements. These predictions help to optimize maintenance costs and avoid critical downtime.
 - Key industries: Automotive, manufacturing, logistics, and oil and gas
- **Real-time personalization:** Generate personalized recommendations for customers in real time.
 - Key industries: Retail and e-commerce.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that you can use to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

In the world of machine learning and custom model training and deployment, you should consider implementing more governance and adopt practices for operations like [MLOps](#), [DevOps](#), and continuous integration/continuous delivery (CI/CD).

Reliability

Reliability ensures that your application can meet the commitments that you make to your customers. For more information, see [Overview of the reliability pillar](#).

Most of the components that are used in this example scenario are managed services that automatically scale. The [availability of those services](#) varies by region.

Apps that are based on machine learning usually require one set of resources for training and another for serving. Resources that are required for training generally don't need high availability, because live production requests don't directly hit these resources. However, resources that are required for serving production requests need high availability.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

Azure pricing: First-party services that provide infrastructure as a service (IaaS) and platform as a service (PaaS) on Azure use a consumption-based pricing model. They don't require a license or subscription fee. You can use the [Azure Pricing Calculator](#) to estimate the cost of using the services with your specific data size and workloads.

See the following links for more Azure service pricing resources:

- [Data Lake Storage](#)
- [Machine Learning](#)

Power Platform pricing: Power Apps and Power Automate, provided as software as a service (SaaS) have their own pricing models, including per app plan and per user.

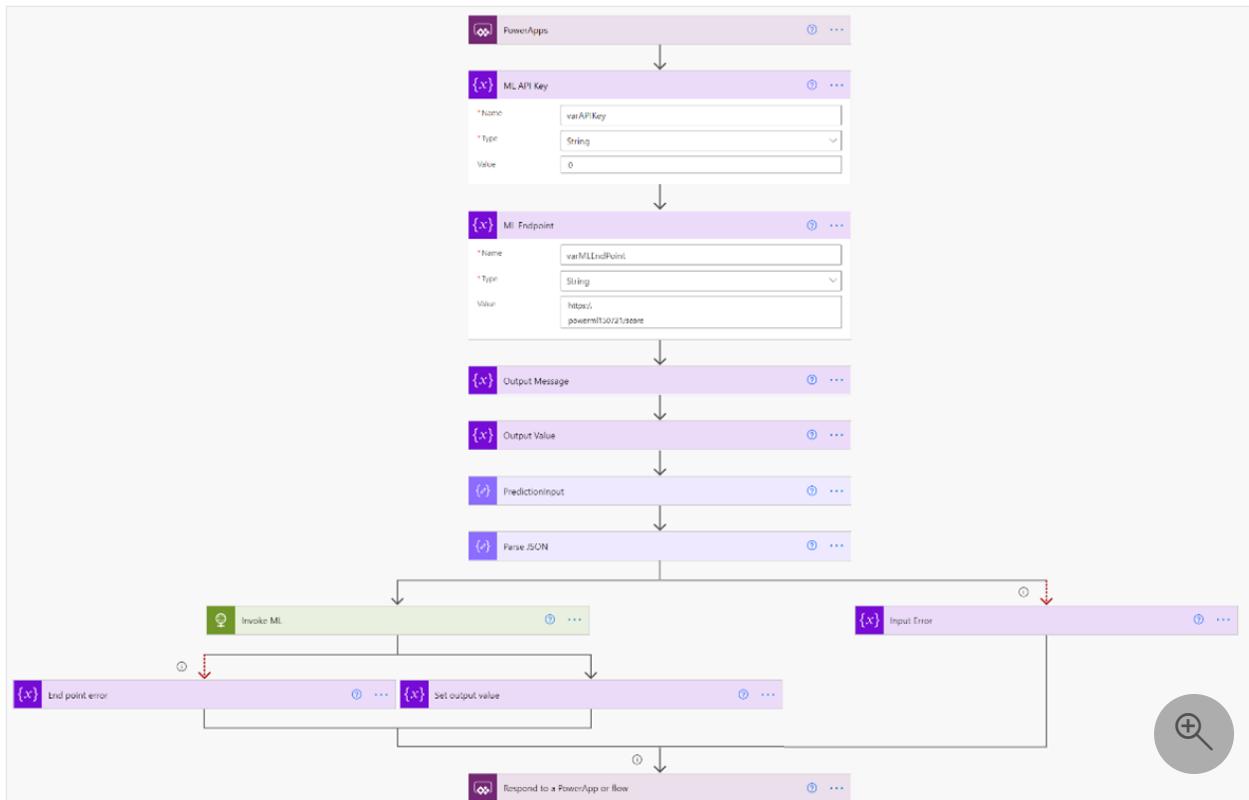
- [Power Apps](#)
- [Power Automate](#)

Deploy this scenario

Here's an example user interface for the app, which was created in Power Apps by using the low-code interface that Power Apps provides.



You can use Power Automate to build a low-code workflow to parse a user's input, pass it to the Machine Learning endpoint, and retrieve the prediction. For more information, see [Tutorial: Consume Azure Machine Learning models in Power BI](#).



To deploy this end-to-end example, follow step by step instructions in [Car Price Predictor - Azure ML + Power App Solution](#).

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Christina Skarpathiotaki](#) | AI Cloud Solution Architect

Other contributors:

- [Brady Leavitt](#) | Technical Specialist, AI/ML
- [Jason Martinez](#) | Technical Writer

To see non-public LinkedIn profiles, sign in to LinkedIn.

Next steps

- Build intelligent applications infused with world-class AI
- Create and attach an Azure Kubernetes Service cluster
- Create machine learning models
- Data in Azure Machine Learning

- Deploy a model to Azure Container Instances with CLI (v1)
- Deploy and score a machine learning model by using an online endpoint
- How Azure Machine Learning works: resources and assets (v2)
- Introduction to data for machine learning
- Introduction to machine learning operations (MLOps)

Related resources

- Analytics end-to-end with Azure Synapse
- Artificial intelligence (AI) architecture design
- Compare the machine learning products and technologies from Microsoft
- End-to-end computer vision at the edge for manufacturing
- Machine learning operations (MLOps) framework to upscale machine learning lifecycle with Azure Machine Learning

Azure and Microsoft 365 scenarios

Article • 10/10/2023

Microsoft 365 is a suite of apps that help you stay connected and get things done. It includes these tools:

- [Word](#). Create documents and improve your writing with built-in intelligent features.
- [Excel](#). Simplify complex data and create easy-to-read spreadsheets.
- [PowerPoint](#). Easily create polished presentations.
- [Teams](#). Bring everyone together in one place to meet, chat, call, and collaborate.
- [Outlook](#). Manage your email, calendar, tasks, and contacts in one place.
- [OneDrive](#). Save, access, edit, and share files.
- [Exchange](#). Work smarter with business-class email and calendaring.
- [SharePoint](#). Share and manage content, knowledge, and applications to enhance teamwork, make information easy to find, and collaborate across your organization.
- [Access](#). Create database apps easily in formats that best serve your business.
- [Publisher](#). Create professional layouts and publish content in a way that suits your audience.
- [Intune](#). Secure, deploy, and manage all users, apps, and devices without disrupting your processes.

This article provides a summary of architectures and solutions that use Azure together with Microsoft 365.

Watch this short video to learn how Microsoft 365 apps and services can help your organization work, learn, connect, and create:

https://www.youtube-nocookie.com/embed/d6p_aKM1M3o

Solutions across Azure and Microsoft 365

The following articles provide detailed analysis of solutions that feature integration between Azure and Microsoft 365.

Microsoft 365 (general)

Architecture	Summary	Technology focus
Microsoft Entra security for AWS	Learn how Microsoft Entra ID can help secure and protect Amazon Web Services (AWS) identity management and account access. If you already use Microsoft Entra ID for Microsoft 365, this solution is easy to deploy.	Identity
Defender for Cloud Apps and Microsoft Sentinel for AWS	Learn how Microsoft Defender for Cloud Apps and Microsoft Sentinel can help secure and protect AWS account access and environments. If you already use Microsoft Entra ID for Microsoft 365, this solution is easy to deploy.	Security
Manage Microsoft 365 tenant configuration with Azure DevOps	Learn how to manage Microsoft 365 tenant configuration by using Microsoft365DSC and Azure DevOps.	Web
Power Automate deployment at scale	Learn how to use a hub-and-spoke architectural model to deploy Power Automate parent and child flows. This architecture is for Power Automate workflows that replace SharePoint 2010 workflows, and for new SharePoint Online sites.	Integration
Virtual health on Microsoft Cloud for Healthcare	Learn how to develop a virtual health solution by using Microsoft Cloud for Healthcare. Microsoft Cloud for Healthcare brings together capabilities from Microsoft 365, Azure, and other technologies to help healthcare organizations create fast, efficient, and highly secure healthcare solutions.	Web

Excel

Architecture	Summary	Technology focus
Interactive price analytics	Use transaction history data to develop a pricing strategy that shows how the demand for your products responds to your prices.	Analytics

Exchange Online

Architecture	Summary	Technology focus
Enhanced-security hybrid messaging—client access	Use Microsoft Entra multifactor authentication to enhance your security in a client access scenario that uses Exchange.	Hybrid
Enhanced-security hybrid messaging—mobile access	Use Microsoft Entra multifactor authentication to enhance your security in a mobile access scenario that uses Exchange.	Hybrid
Enhanced-security hybrid messaging—web access	Use Microsoft Entra multifactor authentication to enhance your security in a web access scenario that uses Exchange.	Hybrid

SharePoint

Architecture	Summary	Technology focus
Enterprise-scale disaster recovery	Learn about a large-enterprise architecture for SharePoint, Dynamics CRM, and Linux web servers that runs on an on-premises datacenter and fails over to Azure infrastructure.	Management/Governance
Highly available SharePoint farm	Deploy a highly available SharePoint farm that uses Microsoft Entra ID, a SQL Server Always On instance, and SharePoint resources.	Web
Hybrid SharePoint farm with Microsoft 365	Deliver highly available intranet capability and share hybrid workloads with Microsoft 365 by using SharePoint servers, Microsoft Entra ID, and SQL Server.	Web
SharePoint farm for development testing	Deploy a SharePoint farm for development testing. Use Microsoft Entra ID, SQL Server, and SharePoint resources for this agile development architecture.	DevOps

Teams

Architecture	Summary	Technology focus
Governance of Teams guest users	Learn how to use Teams and Microsoft Entra entitlement management to collaborate with other organizations while maintaining control over resource use.	Identity

Architecture	Summary	Technology focus
Provide security for your Teams channel bot and web app behind a firewall	Provide security for the connection to a Teams channel bot's web app by using Azure Private Link and a private endpoint.	Security
Teacher-provisioned virtual labs in Azure	Learn how you can use Azure Lab Services to set up identical VMs from templates for use in training, customer demos, and software development. Students and stakeholders can access the VMs via Teams integration.	DevOps

Related resources

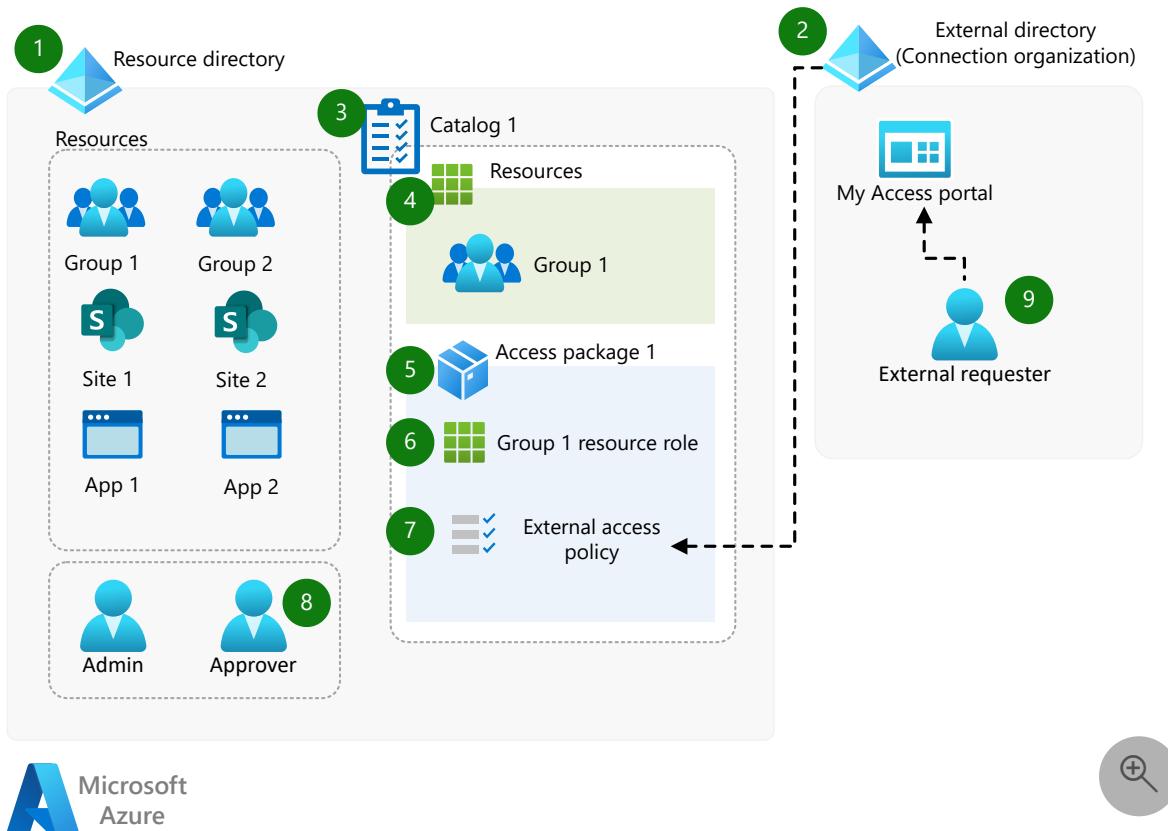
- [Browse our Microsoft 365 architectures](#)

Governance of Microsoft Teams guest users

Microsoft Entra ID Microsoft 365 Microsoft Teams

This example scenario helps users collaborate with other organizations, by providing identity and governance controls for external users when using Microsoft Entra B2B collaboration.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

- 1. Resource directory** - This is the Microsoft Entra directory that contains resources, which are Microsoft 365 groups and teams. For this example, the resource is a project team that is added to the access package, so that users external to the organization can request access to it.

2. **External directory (Connected organization)** -This is the external Microsoft Entra directory that contains external users from the connected organization. These users can be allowed by a policy to request access to the project team.
3. **Catalog 1** - A catalog is a container for related resources and access packages. Catalog 1 contains the project team and its access package.

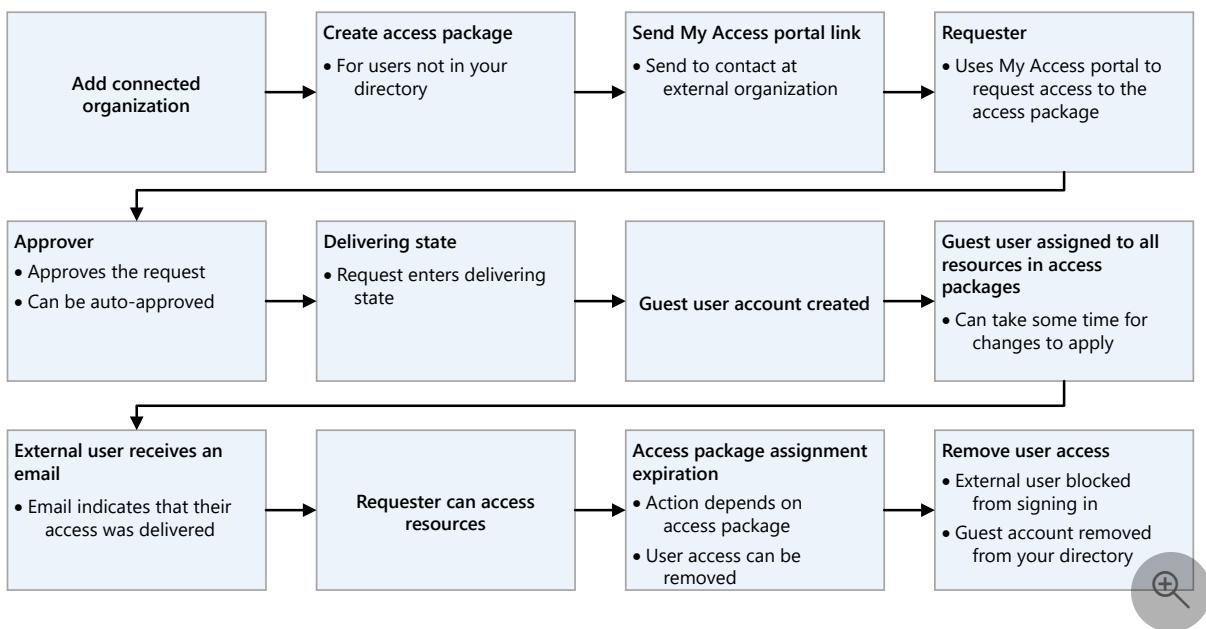
Catalogs allow for delegation, so that non-administrators can create access packages. Catalog owners can add resources that they own to a catalog.
4. **Resources** - These are the resources that appear in the access packages. They can include security groups, applications, and SharePoint Online sites. In this example, it's the project team.
5. **Access 1** - An access package is a collection of resources with access types for each. Access packages are used to govern access for internal and external users. In this example, the project team is the resource with a single policy that allows external users to request access. Internal users in this example don't need to use Microsoft Entra entitlement management. They're added to the project team by using Microsoft Teams.
6. **Group 1 resource role** - Resource roles are permissions that are associated with, and defined by, a resource. A group has two roles—member and owner. SharePoint sites typically have three roles, but can have additional custom roles. Applications can have custom roles.
7. **External access policy** - This is the policy that defines the rules for assignment to an access package. A policy is used in this example to ensure that users from connected organizations can request access to the project team. After a request is made, approval is required from approvers as defined in the policy. The policy also specifies time limits and renewal settings.
8. **Approver** - An approver approves the access request. This can be an internal or external user.
9. **Requester** - This is the external user that requests access via the My Access portal. The portal only shows the access packages that the requester is allowed to request.

Requesting access to a resource for users external to the organization flow

Here is a high-level workflow that shows how access to the Microsoft 365 group or team is granted to external users. It includes the removal of a guest account when access is no

longer required or a time limit is reached.

How access works for external users



Components

- Microsoft Entra ID [↗](#) offers cloud-based identity and access management services that provide a way for users to sign in and access resources. It has the following features and capabilities:
 - Microsoft Entra entitlement management** is an identity governance feature that enables organizations to manage identity and access lifecycles at scale, by automating access request workflows, access assignments, reviews, and expiration.
 - Microsoft Entra business-to-business (B2B) collaboration** is used by Microsoft Entra entitlement management to share access, so that internal users can collaborate with external users.
 - Microsoft Entra access reviews** enable organizations to efficiently manage group memberships, access to enterprise applications, and role assignments. User access can be reviewed on a regular basis to make sure only the right people have continued access.
 - Microsoft Teams guest access** allows external users to access teams, channels, resources, chats, and applications, while you maintain control over your corporate resources.
 - Microsoft Entra Conditional Access** brings signals together to make decisions, and enforce organizational policies. Conditional Access in this solution was used to enforce Terms of Use agreements and multifactor authentication, and to set session timeouts for guest accounts.

Alternatives

An alternative solution to Microsoft Entra entitlement management is to allow internal users to invite external users to a team. An invited user could then create a guest account in the resource directory.

This alternative doesn't provide the identity and governance controls that the customer required. The deficiencies compared to AD entitlement management are:

- External users can't request access—there must be an invitation. The external user has to know how to request an invitation, a process that can vary by team, and change over time.
- There are no business justifications, email notifications, review processes, or approval processes, which is an auditing issue.
- Access to specific resources can't be managed, removed, or updated easily. Because project resources are likely to change, this is an efficiency issue.
- If an external user is invited but the user's organization isn't allowed, the user is denied access, causing confusion.
- Guest accounts aren't automatically removed and there's no expiration set. A manual process to handle expiration is error-prone and less efficient than an automatic process that requires limits to be set at account creation. This is a security issue and an efficiency issue.

Building a custom solution to handle these issues is unlikely to be cost-competitive or feature-competitive with AD entitlement management.

Scenario details

This example scenario was built during the COVID-19 pandemic, when a customer had an immediate requirement to collaborate with other organizations. This meant providing identity and governance controls for external users.

Microsoft Teams was the customer's primary tool for company communications. Users collaborated by using Teams chat, meetings, and calling. Teams channels provided them access to files and conversations.

Teams meetings provided an effective way to meet with external users. However, external users couldn't access the teams and channels, so collaborating with them was clumsy, and productivity was impeded. The customer needed something better.

Teams provides two options to communicate and collaborate with external users:

- **External access** - A type of federation that allows internal users to find, call, and chat with external users. External access users can't be added to teams unless they are invited as guests by using guest access.
- **Guest access** - Allows internal users to invite external users to join a team. The invited users get a guest account in Microsoft Entra ID. Guest access allows external users to be invited to teams and provides access to documents in channels, and to resources, chats, and applications. The customer maintains control over corporate data as required.

Guest access met the customer's collaboration requirements, but gave rise to security and governance concerns:

- Guests must only have access to specific teams as required, and only for as long as necessary. When a project completes, the guest account must be removed.
- There must be an approval process for creating guest accounts that satisfies auditing requirements. Internal users must review requests and approve them as appropriate.
- It must be possible to build and automate the solution quickly. No guest accounts can be created until appropriate security and governance controls are in place.

Microsoft Entra entitlement management was the primary tool to satisfy the security and governance requirements:

- It helps efficiently manage access to Microsoft 365 groups, including teams, applications, and SharePoint online sites, for both internal and external users.
- It provides the ability to automate access request workflows, access assignments, reviews, and expiration.

Guest access and Microsoft Entra entitlement together met the customer's collaboration requirements. External users can join selected teams and the access is managed. Moreover, Microsoft Entra entitlement management offers functionality for possible future use, such as managing access to resources other than teams.

Potential use cases

This solution applies to any situation that requires managing access—for those internal and external users that need it, to groups, applications, and SharePoint Online sites. Microsoft Entra entitlement management has these features and advantages:

- There's simplified onboarding and management for employee access to resources such as:
 - Microsoft Entra security groups.
 - Microsoft 365 groups.

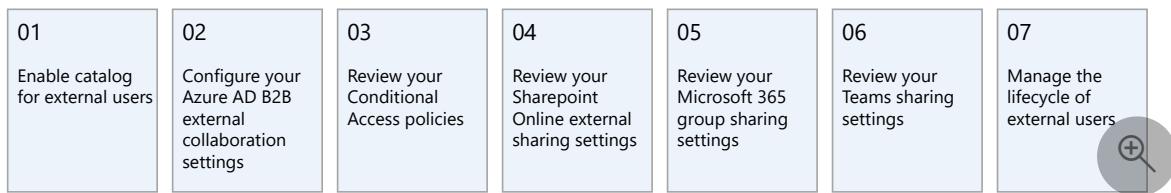
- Microsoft 365 teams.
- Applications, including SaaS applications.
- Custom applications that implement appropriate security measures.
- SharePoint Online sites.
- There are simplified procedures for external users to gain access to the resources that they need.
- You can designate which connected organizations are allowed to provide external users that can request access.
- A user who requests access, and is approved, is automatically invited into the team directory, and assigned access to resources.
- A time limit can be set on a user's access to resources, with automatic removal when the limit is reached.
- When access expires for an external user that has no other access package assignments, the user's account can be automatically removed.
- You can ensure that users have no more access than they require.
- There's an approval process for access requests that includes approval by designated individuals, such as managers.
- You can manage access to other resources that rely upon Microsoft Entra security groups or Microsoft 365 groups. An example is granting licenses to users by using group-based licensing.
- You can delegate to non-administrators the ability to create access packages that contain resources that users can request.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

An important implementation step is configuring tenant settings to allow for external users.

Change tenant settings for external users first



1. **Enable catalog for external users** - Make sure the catalog has **Enabled for external users** set to **Yes**. By default, when you create a new catalog in Microsoft

Entra entitlement management, it's enabled to allow external users to request access to packages in the catalog.

2. Microsoft Entra B2B external collaboration settings - The Azure B2B external collaboration settings can affect whether you can use Microsoft Entra entitlement management to invite external users to resources. Verify these settings:

- Check whether guests are allowed to invite other guests to your directory. We recommend setting **Guests can invite** to **No** to only allow governed invitations.
- Make sure that you're allowing or blocking invitations appropriately. For more information, see [Allow or block invitations to B2B users from specific organizations](#).

3. Review your Conditional Access policies - Verify Conditional Access to make sure guest users are excluded from any Conditional Access policies that they can't satisfy. Otherwise they can't sign in to your directory and won't have access to the resource.

4. Review your SharePoint Online external sharing settings - If you include SharePoint Online sites in an access package for external users, make sure that you configure the organization-level external sharing setting. Set as **Anyone** if sign-in isn't required, or **Existing guests** for invited users. For more information, see [Change the organization-level external sharing setting](#).

5. Review your Microsoft 365 group sharing settings - If you include Microsoft 365 groups or teams in an access package for external users, make sure that **Let users add new guests to the organization** is set to **On** to allow guest access.

6. Review your Teams sharing setting - If you include teams in an access package for external users, make sure that **Allow guest access in Microsoft Teams** is set to **On** to allow guest access. Also, check that the Teams **Guest Access settings** are configured.

7. Manage the lifecycle of external users - You can select what happens when an external user no longer has any access package assignments. This happens when all assignments are either relinquished by the user, or expired. By default, the user is blocked from signing in to your directory. After 30 days, the guest user account is removed from your directory.

Additional considerations:

- **Access assignment** - Access packages don't replace other mechanisms for access assignment. They're most appropriate in situations such as:
 - Employees need time-limited access for a particular task.
 - Access requires the approval of a manager or other designated individual.
 - Departments want to manage their resources without IT involvement.

- Two or more organizations are collaborating on a project, so multiple users from one organization need to be invited to access another organization's resources.
- **Updating resources** - With Microsoft Entra entitlement management, you can change the resources in an access package at any time. The users of the package have their resource access automatically adjusted to match the changed package.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

- The use of Microsoft Entra entitlement management requires a Microsoft Entra ID P2 license.
- Guest access can be used with all Microsoft 365 Business Standard, Microsoft 365 Business Premium, and Microsoft 365 Education subscriptions. No additional Microsoft 365 license is necessary.
- The billing model for Microsoft Entra External ID applies to guests in Microsoft 365. Only external users can be invited as guests.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Martin Boam](#) | Associate Architect

Next steps

- [Overview of teams and channels in Microsoft Teams](#)
- [Understand teams and channels in Microsoft Teams](#)
- [Use guest access and external access to collaborate with people outside your organization](#)
- [Create a new access package in Microsoft Entra entitlement management](#)
- [Tutorial: Manage access to resources in Microsoft Entra entitlement management](#)
- [What is Microsoft Entra entitlement management?](#)
- [What is Microsoft Entra ID Governance?](#)
- [What are Microsoft Entra access reviews?](#)

- Common scenarios in Microsoft Entra entitlement management
- Govern access for external users in Microsoft Entra entitlement management
- Govern access for users external your organization
- Collaborate with guests in a team
- Use guest access and external access to collaborate with people external your organization
- Collaborating with people outside your organization
- What is Conditional Access?

Related resources

- Create an AD DS resource forest in Azure
- Deploy AD DS in an Azure virtual network
- Hybrid identity
- Integrate on-premises AD domains with Microsoft Entra ID
- Microsoft Entra identity management and access management for AWS

Hybrid SharePoint farm with Microsoft 365

Microsoft Entra ID SQL Server Windows Server SharePoint Microsoft 365

💡 Solution ideas

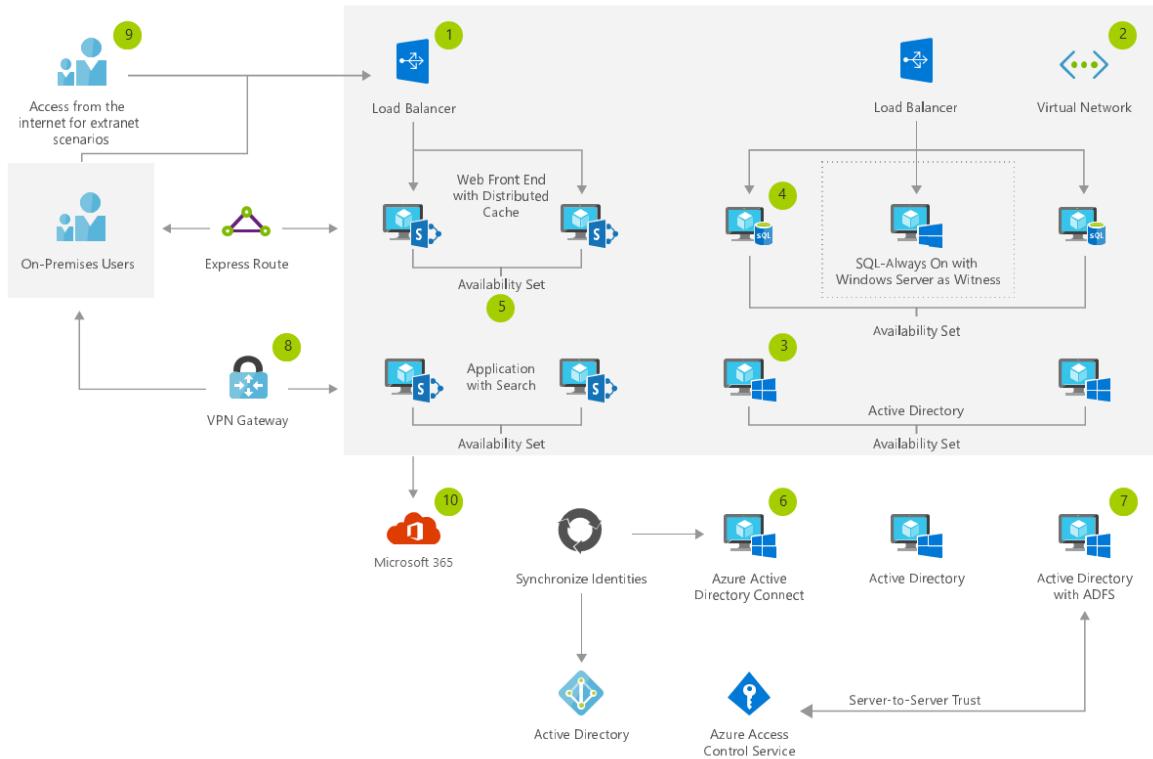
This article is a solution idea. If you'd like us to expand the content with more information, such as potential use cases, alternative services, implementation considerations, or pricing guidance, let us know by providing [GitHub feedback](#).

This solution provides a highly available deployment of SharePoint, by using a load-balanced Microsoft Entra instance, a highly available SQL always-on instance, and highly available SharePoint resources.

Potential use cases

This solution addresses the need to deliver a highly available intranet capability, by using the latest and greatest supported platforms.

Architecture



[Download an SVG of this architecture.](#)

Dataflow

1. Create a resource group to host all Azure based infrastructure and services.
2. Create a virtual network in Azure.
3. Deploy Windows Servers to host Active Directory services for SharePoint, SQL server service accounts, and machine accounts.
4. Deploy SQL Server Always-on for high availability (HA) support for the SharePoint farm.
5. Deploy the SharePoint Server instances. In this scenario, we use two frontend servers with distributed cache and two applications with search roles. This gives us high availability.
6. Install Microsoft Entra Connect on an on-premises server, to synchronize your identities to Microsoft Entra ID.
7. Optionally configure Active Directory Federation Services on premises, to support federated authentication to Microsoft 365.
8. Deploy ExpressRoute or set up a site-to-site VPN link, for administrative access to the servers that are hosted in Azure VMs.
9. Set up and provision external access to the hybrid farm that's hosted in Azure VMs.
10. Set up and configure hybrid workloads between Microsoft 365 and the SharePoint farm.

Components

- [Azure Resource Group](#): Container that holds related resources for an Azure solution.
- [Virtual Network](#): Provision private networks, and optionally connect to on-premises datacenters.
- [Storage Accounts](#): Enable durable, highly available, and massively scalable cloud storage.
- [Microsoft Entra ID](#): Synchronize on-premises directories, and enable single sign-on.
- [SharePoint Server](#): Microsoft's collaboration server product.
- Host enterprise [SQL Server](#) apps in the cloud.
- [Load Balancer](#): Deliver high availability and network performance to your applications.
- [Azure ExpressRoute](#): Dedicated private network fiber connections to Azure
- [VPN Gateway](#): Establish secure, cross-premises connectivity.
- Microsoft Entra Connect: Synchronize on-premises directories, and enable single sign-on.
- Active Directory Federation Services: Synchronize on-premises directories, and enable single sign-on.
- Hybrid Workloads: Scale between on-premises environments and the cloud.

Next steps

- [Azure Resource Group Documentation](#)
- [Virtual Network Documentation](#)
- [Storage Documentation](#)
- [Active Directory Documentation](#)
- [SharePoint Server Documentation](#)
- [SQL Server Documentation](#)
- [Load Balancer Documentation](#)
- [ExpressRoute Documentation](#)
- [VPN Gateway Documentation](#)
- [Microsoft Entra Connect Documentation](#)
- [Active Directory Federation Services Documentation](#)
- [Hybrid Workloads Documentation](#)

Manage Microsoft 365 tenant configuration by using Microsoft365DSC and Azure DevOps

Azure DevOps

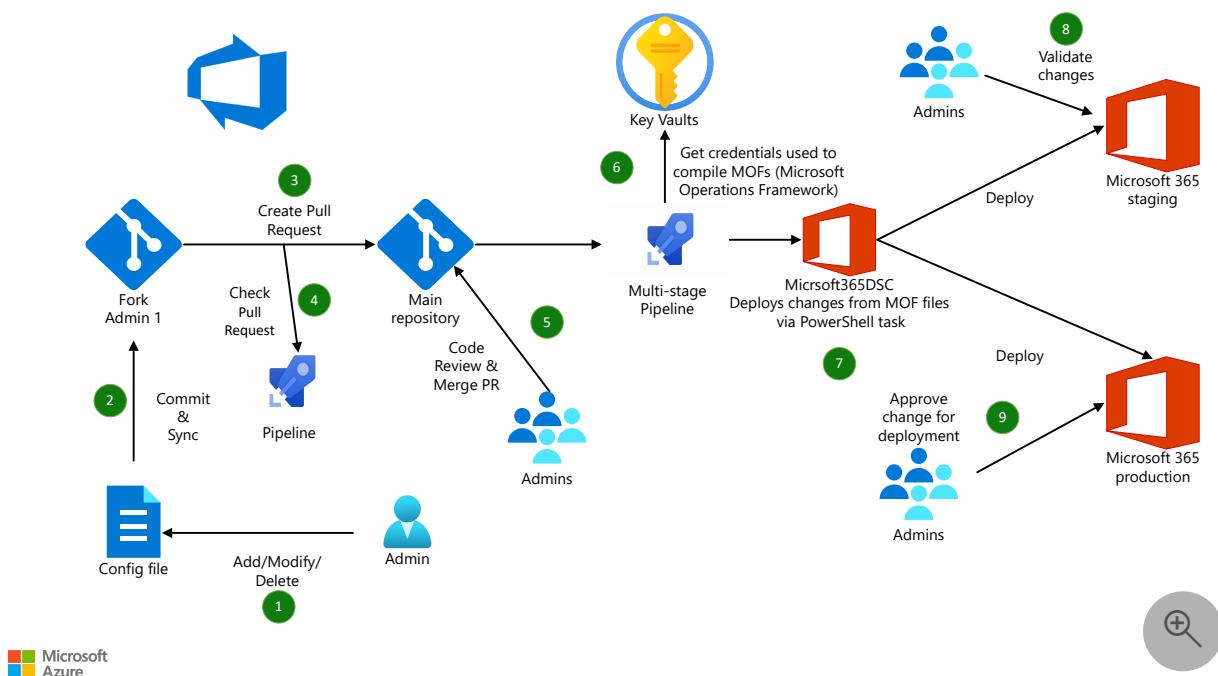
Azure Key Vault

Azure Windows Virtual Machines

Microsoft 365

The solution described here tracks changes made by service administrators and adds an approval process to deployments to Microsoft 365 tenants. It can help you prevent untracked changes to Microsoft 365 tenants. It also helps to prevent configuration drift between multiple Microsoft 365 tenants.

Architecture



Download a [Visio file](#) of this architecture.

Workflow

1. Admin 1 adds, updates, or deletes an entry in Admin 1's fork of the Microsoft 365 config file.
2. Admin 1 commits and syncs the changes to Admin 1's forked repository.
3. Admin 1 creates a pull request (PR) to merge the changes to the main repository.

4. The build pipeline runs on the PR.
5. Admins review code and merge the PR.
6. The merged PR triggers a pipeline to compile Managed Object Format (MOF) files.
The pipeline calls Azure Key Vault to retrieve credentials that are used in the MOFs.
7. An Azure PowerShell task in a multistage pipeline uses the compiled MOF files to deploy configuration changes via Microsoft365DSC.
8. Admins validate changes in a staged Microsoft 365 tenant.
9. Admins get notification from the approval process in Azure DevOps for the production Microsoft 365 tenant. Admins approve or reject the changes.

Components

- [Azure Pipelines](#) enables continuous integration (CI) and continuous delivery (CD) to test and build your code and ship it to any target.
- [Azure Key Vault](#) improves the security of storage for tokens, passwords, certificates, API keys, and other secrets. It also provides tightly controlled access to these secrets.
- [Microsoft365DSC](#) provides automation for the deployment, configuration, and monitoring of Microsoft 365 tenants via PowerShell DSC.
- [Windows PowerShell DSC](#) is a management platform in PowerShell. You can use it to manage your development infrastructure by using a configuration-as-code model.

Alternatives

As a next step, you can use DSC in [Azure Automation](#) to store configurations in a central location and add reporting of compliance with the desired state.

This architecture uses Key Vault to store Azure App Service certificates or user credentials that are used for authentication to the Microsoft 365 tenant. Key Vault provides scalability. As an alternative, you can use pipeline variables to reduce the complexity of the solution.

Scenario details

Many companies are adopting DevOps practices and want to apply these practices to their Microsoft 365 tenants. If you don't adopt DevOps for Microsoft 365, you might encounter some common problems:

- Misconfiguration
- Challenges with tracking configuration changes

- No approval process for tenant modifications

You can use the solution described in this article to automate changes to Microsoft 365 tenant configurations by using [Azure DevOps](#) and [Microsoft365DSC](#).

Microsoft365DSC is a [PowerShell Desired State Configuration \(DSC\)](#) module. You can use it to configure and manage Microsoft 365 tenants in a true DevOps style: configuration as code.

Potential use cases

This solution can help you manage Microsoft 365 tenant configuration in a controlled and automated way, using DevOps tools and practices, across:

- Development, test, acceptance, and production environments.
- Multiple customer tenants, as in a managed-service provider scenario.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload.

For more information, see [Microsoft Azure Well-Architected Framework](#).

Most people starting out with PowerShell DSC find that it takes a while to learn it. It helps if you have a solid understanding of PowerShell and experience with creating scripts.

Operations

Some operations teams consider Azure DevOps to be a tool for developers. But these teams can benefit from using Azure DevOps. Operations teams can:

- Store their scripts in a repository and add source control and versioning.
- Automate deployments of scripts.
- Use boards to track tasks, projects, and more.

Using a configuration-as-code model isn't a one-time task. It's a shift in your way of working and a fundamental change for all team members. You no longer make changes manually. Instead, everything is implemented in scripts and deployed automatically. All team members need to have the skills to make this change.

Scalability

You can use this solution when you're working with multiple environments, multiple workloads, and/or multiple teams. You can configure the validation process so that experts need to approve each workload. You can also extend the solution to deploy to multiple tenants, for a dev/test/acceptance/production scenario and/or for multiple organizations.

To increase scalability even further, you can use an aggregated-configuration data solution like [Datum](#).

Security

Most Microsoft365DSC resources support authentication via user name and password. But we don't recommend that type of authentication because Microsoft best practices recommend multifactor authentication. Application credentials is the preferred method, where supported by the Microsoft 365 resources. For example, SharePoint Online, Microsoft Entra ID, and other resources support application credentials.

If you build a Microsoft365DSC solution on Azure DevOps, you can also take advantage of the security in [Azure Pipelines](#) and an [approval process](#) to safeguard deployment to your production tenant.

DevOps

You can run this solution in Azure DevOps Server. You could create a similar solution in GitHub by using GitHub Actions.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

For Azure DevOps pricing information, see [Pricing for Azure DevOps](#). If you incorporate Key Vault into your solution, you can find [pricing information here](#).

You can also use the [Azure pricing calculator](#) to estimate costs.

Contributors

This article is being maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Derek Smay ↗](#) | Senior Customer Engineer

Next steps

- [Managing Microsoft 365 in true DevOps style with Microsoft365DSC and Azure DevOps ↗](#)
- [Microsoft365DSC source code ↗](#)
- [Microsoft365DSC YouTube channel ↗](#)
- [Microsoft365DSC site ↗](#)
- [Microsoft365DSC export generator tool ↗](#)

Related resources

- [End-to-end governance in Azure when using CI/CD](#)
- [CI/CD for Windows desktop apps](#)
- [DevOps Checklist](#)

Real-time collaboration with Azure and Microsoft 365

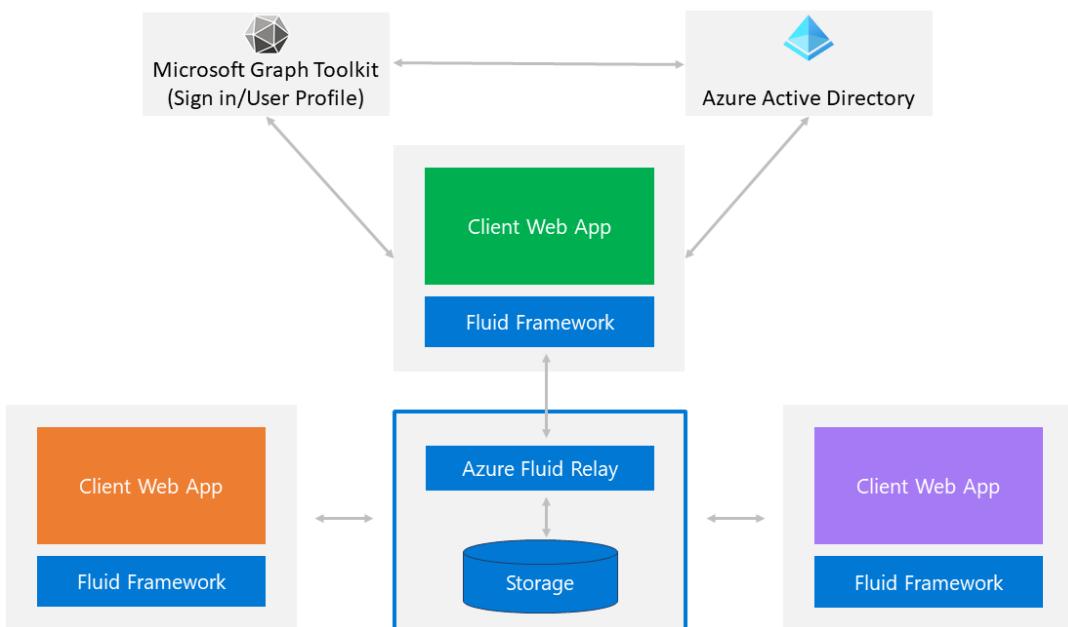
Microsoft Entra ID Microsoft Graph Azure App Service Azure Fluid Relay Microsoft 365

💡 Solution ideas

This article is a solution idea. If you'd like us to expand the content with more information, such as potential use cases, alternative services, implementation considerations, or pricing guidance, let us know by providing [GitHub feedback](#).

This solution shows how you can use libraries and Azure services to meet custom collaboration requirements. In addition to real-time collaboration, this solution supports user presence status. Users can work together in the custom app to collect ideas, see when new ideas are added, modified, or deleted in real time, and avoid data conflicts during collaboration sessions.

Architecture



Download a [PowerPoint file](#) of this architecture.

Dataflow

- An application uses the Login component of the Microsoft Graph Toolkit to enable a user to sign in. The Login component uses the MSAL provider to validate the user's credentials against Microsoft Entra ID.
- After the user signs in, the client web app uses Fluid Framework to connect to Fluid Relay and creates a collaboration session.
- More users sign in and participate in the collaboration session. Fluid Framework merges the data sent and received in each client to ensure that it's synchronized for all users. The [total order broadcast algorithm and eventual consistency](#) are used to ensure this synchronization.
- As users continue to collaborate, Fluid Relay automatically stores the collaboration data.
- As new users join the collaboration session, each user's client retrieves previously stored data and ensures that the user is synchronized with other users in the session.

Components

- [Fluid Framework](#) is a collection of client libraries for distributing and synchronizing shared state. These libraries allow multiple clients to simultaneously create and operate on shared data structures by using coding patterns similar to those used to work with local data.
- [Fluid Relay](#) is a managed service of Fluid Framework. It helps developers build real-time collaborative experiences and replicate state across connected JavaScript clients in real time.
- [Microsoft Entra ID](#) is the Microsoft cloud-based identity and access management service that helps your employees sign in and access resources.
- [Microsoft Graph Toolkit](#) is a collection of reusable, framework-agnostic components and authentication providers for accessing and working with Microsoft Graph.
- [Azure Static Web Apps](#) is a service that automatically builds and deploys full stack web apps to Azure from a code repository.

Scenario details

Collaboration is critical to business efficiency and productivity. Tools like Microsoft Teams provide a great way to collaborate via chat, audio, and video. Word, Excel, and PowerPoint online make it easy to collaborate on various types of documents and spreadsheets with colleagues and customers around the world.

You can use Azure services to add real-time collaborative functionality to custom applications just as you do with off-the-shelf solutions. This solution shows how you can use libraries and Azure services to meet custom collaboration requirements. In addition to real-time collaboration, this solution supports user presence status. Users can work together in the custom app to collect ideas, see when new ideas are added, modified, or deleted in real time, and avoid data conflicts during collaboration sessions.

To meet these requirements, the solution uses Fluid Framework and Azure Fluid Relay. It authenticates users against Microsoft Entra ID by using the Login component of the Microsoft Graph Toolkit and the Microsoft Authentication Library (MSAL) provider.

Potential use cases

This solution applies to companies that build custom application solutions that require:

- Secure application access.
- Real-time data collaboration among multiple users.
- Built-in collaboration data-storage capabilities.

Deploy this scenario

Get a [code sample](#) that demonstrates this scenario.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Dan Wahlin](#) | Principal Cloud Developer

Next steps

For more information about the technologies used in this solution, see these articles:

- [Microsoft Entra ID](#)
- [Azure Fluid Relay](#)
- [Azure Static Web Apps](#)
- [Fluid Framework](#)
- [Microsoft Graph](#)

- Microsoft Graph Toolkit
- Total order broadcast and eventual consistency in Fluid Framework ↗

Related resources

- Enhanced-security hybrid messaging infrastructure—web access
- Enhanced-security hybrid messaging infrastructure—mobile access
- Manage Microsoft 365 tenant configuration by using Microsoft365DSC and Azure DevOps
- Power Automate deployment at scale

Real-time presence with Microsoft 365, Azure, and Power Platform

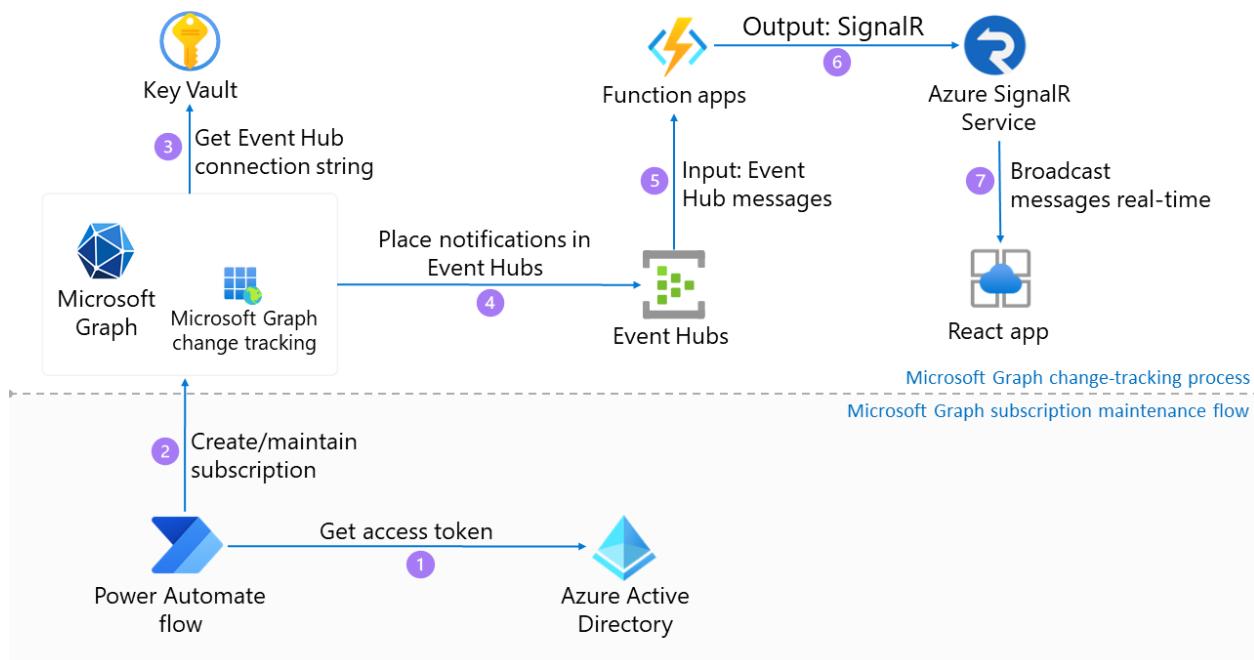
Azure Functions Microsoft Graph Microsoft Power Platform Azure SignalR Service Azure Event Hubs

💡 Solution ideas

This article is a solution idea. If you'd like us to expand the content with more information, such as potential use cases, alternative services, implementation considerations, or pricing guidance, let us know by providing [GitHub feedback](#).

This solution provides presence for a JavaScript front-end application. It uses Microsoft Graph and Microsoft Entra ID to provide real-time presence information. It also uses Power Automate, Azure Event Hubs, Azure Functions, and Azure SignalR Service.

Architecture



Download a [PowerPoint file](#) of this architecture.

Dataflow

1. A Power Automate flow gets an access token for Microsoft Graph by using Microsoft Entra ID.

2. The Power Automate flow retrieves members of the selected team who are on Microsoft Teams. The flow creates a Microsoft Graph presence API subscription and updates it every hour to track changes in presence for the selected team members.
3. When there's a change in member presence, the Microsoft Graph Change Tracking application policy gets a connection string to Event Hubs from Key Vault.
4. When a Microsoft Graph change notification is processed, the message is placed in Event Hubs.
5. Azure Functions takes the Microsoft Graph messages from Event Hubs as input.
6. Azure SignalR Service is defined as an output in Azure Functions. This allows a browser to subscribe to Azure SignalR Service and receive messages in real time.
7. To receive the presence messages in the browser, an app subscribes to Azure SignalR Service.

Components

- [Microsoft Graph](#) provides a unified programmability model that you can use to access the tremendous amount of data in Microsoft 365, Windows 10, and Enterprise Mobility + Security.
- [Microsoft Entra ID](#) is the Microsoft cloud-based identity and access management service that helps users sign in and access resources.
- [Power Automate](#) helps you automate repetitive manual tasks by recording mouse clicks, keystrokes, and copy-and-paste steps from your desktop.
- [Key Vault](#) is a cloud service for storing and accessing secrets with high security.
- [Event Hubs](#) is a scalable event processing service that ingests and processes large volumes of events and data, with low latency and high reliability.
- [Azure Functions](#) enables you to write event-driven serverless code, maintain less infrastructure, and save money.
 - [Event Hubs trigger for Azure Functions](#) enable you to respond to an event sent to an event hub event stream.
 - [Azure SignalR Service output binding for Azure Functions](#) enables you to send messages by using Azure SignalR Service.
- [Azure SignalR Service](#) simplifies the process of adding real-time web functionality to applications over HTTP.
- [Azure Static Web Apps](#) automatically builds and deploys full stack web apps to Azure from a code repository.

Scenario details

Collaboration tools play a significant role in creating productive workspaces for teams. Microsoft 365 tools like Microsoft Teams, Word, and PowerPoint online bring people together and help them to work more effectively. In addition to supporting real-time changes to documents and data, these tools support real-time presence information. Presence makes it easy to know your teammates' availability and see when they join a collaboration session.

You can also add presence to custom collaboration applications by using Microsoft Cloud services. This solution uses Microsoft Cloud APIs and services to enable real-time presence capabilities in custom applications. As people become available in Microsoft Teams, they can be invited to a collaboration session.

Potential use cases

This solution applies to companies that use custom applications that require:

- Real-time change tracking on Microsoft 365 data.
- Broadcasting of data to the browser in real time.
- Notifications to application users when a person's availability changes.
- A secure way to retrieve secrets that are used in an application.

Deploy this scenario

Get a [code sample](#) that demonstrates this solution.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Dan Wahlin](#) | Principal Cloud Developer

Next steps

For more information about the services and products that are used in this solution, see these articles:

- [Microsoft Graph](#)
- [Microsoft Entra ID](#)

- Power Automate
- Key Vault
- Event Hubs
- Azure Functions
- Azure SignalR Service

Related resources

- Enhanced-security hybrid messaging infrastructure—web access
- Enhanced-security hybrid messaging infrastructure—mobile access
- Manage Microsoft 365 tenant configuration by using Microsoft365DSC and Azure DevOps
- Power Automate deployment at scale

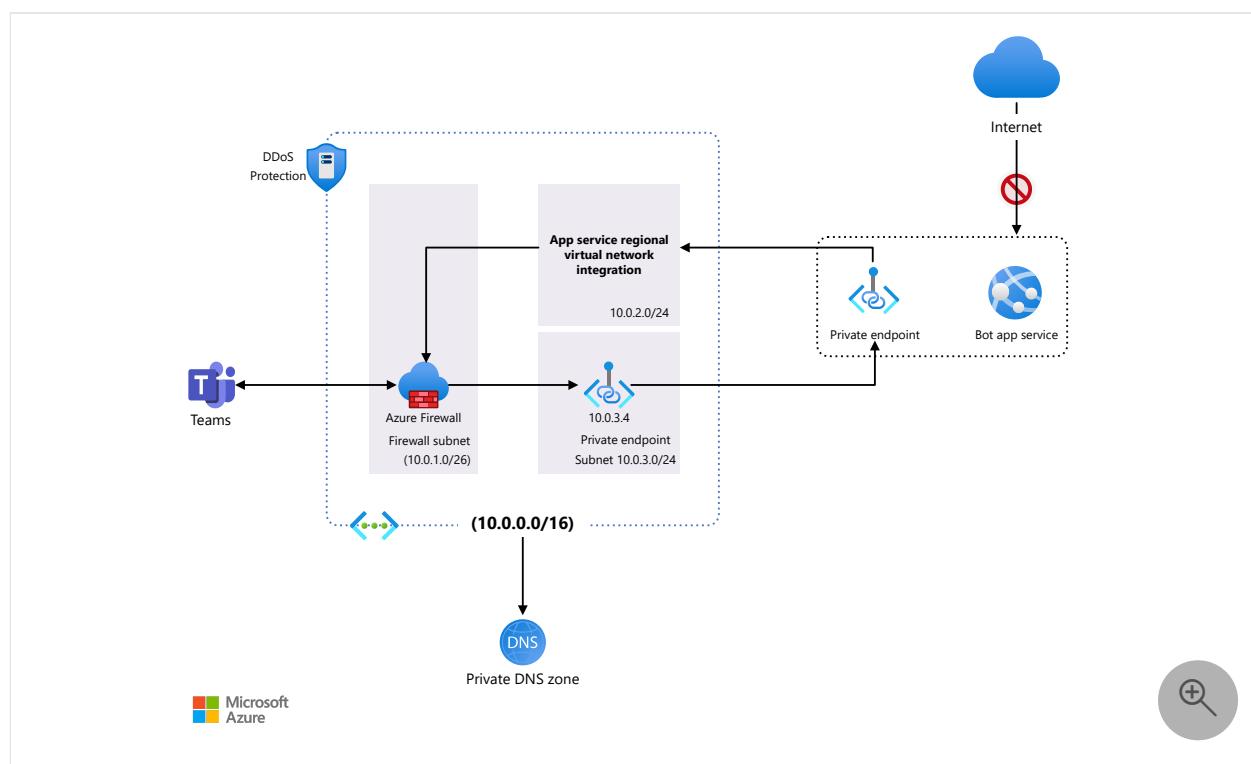
Help secure your Microsoft Teams channel bot and web app behind a firewall

Azure App Service

Azure Web Application Firewall

This example scenario helps secure the connection to a Microsoft Teams channel bot's web app by using Azure Private Link and Azure Private Endpoint. At the same time, it enables channels in the Teams client to communicate with the bot through an IP that's exposed through an Azure Firewall instance.

Architecture



Download a [Visio file](#) of this architecture.

Dataflow

- **Azure Virtual Network** enables communications between Azure resources. The virtual network in this example uses the address space of 10.0.0.0/16, and contains three subnets for use by the scenario's required components:
 - *Azure Firewall Subnet* (10.0.1.0/26).

- *Virtual Network Integration Subnet* (10.0.2.0/24), which is used to route traffic from the bot's private endpoint to the firewall.
 - *Private Endpoint Subnet* (10.0.3.0/24), which is used to route traffic from the firewall to the bot's private endpoint.
- **Azure Firewall** exposes a single public IP address that clients can use to communicate with the underlying bot services. Ordinarily, a firewall is placed in its own virtual network, which is a common pattern for [hub and spoke](#) architectures, but this simplified example deploys all services and resources into a single virtual network. The Azure Firewall instance is placed in its own subnet.
- **Route table** defines the routes that traffic takes within the virtual network. It ensures that traffic coming to and from the bot passes through the firewall.
 - The default route with the 0.0.0.0/0 address prefix instructs Azure to route traffic that isn't within the address prefix of any other route to the subnet where the Azure Firewall instance is deployed. In this example, it's the only route.
 - The *Virtual Network Integration Subnet* and the *Private Endpoint Subnet* are associated with the route table, ensuring that any traffic passing through them is routed through the firewall.
- **Bot Service** consists of the bot [app service plan](#), [app service](#), and [bot channels registration](#).
 - The app service has a registered custom domain that points to the IP address of the firewall. This way, the app service can be accessed only through the firewall.
- **Azure Private Link** service for inbound access to the bot app service over an [Azure private endpoint](#).
- **Virtual network integration** connects the app service to the virtual network, ensuring that outbound traffic from the bot app service passes through the firewall.

Components

- [Virtual Network ↗](#)
- [Azure Firewall ↗](#)
- [Azure Bot Services ↗](#)
- [Azure App Service ↗](#)
- [Azure Private Link ↗](#)

Alternatives

- An [App Service Environment](#) can provide a fully isolated and dedicated environment for securely running App Service apps at high scale. This example doesn't make use of an App Service Environment to reduce costs, but the sample architecture could support it, with modifications.

Scenario details

Bots allow Teams users to interact with web services through text, interactive cards, and task modules. The Microsoft Bot Framework and Azure Bot Services give you an easy-to-use set of tools for creating and managing these bots.

You can develop bots by using a variety of languages, such as C#, JavaScript, and Python. After they're developed, you can deploy them to Azure. A key component of a bot is the web app, which contains the core logic and interface that users communicate with. One of the key requirements for the bot to work is that it must expose a publicly accessible HTTPS endpoint.

InfoSec policy commonly requires that all incoming traffic to web apps go through a corporate firewall. This means that all traffic that goes to a bot, and responses from the bot, must route through a corporate firewall, as with any other web app.

Potential use cases

Organizations can utilize bots for mobile and desktop users. Some examples include:

- Simple queries. Bots can deliver an exact match to a query or a group of related matches to help with disambiguation.
- Multi-turn interactions. By helping anticipate possible next steps, bots make it much easier for people to complete a task flow.
- Reaching out to users. Bots can send a message when something has changed in a document or a work item is closed.

Considerations

Monitoring

Although monitoring isn't implemented in this example scenario, a bot's app service can utilize [Azure Monitor](#) services to monitor its availability and performance.

Scalability

The bots used in this scenario are hosted on Azure App Service. As a result, you can use the standard App Service autoscaling features to automatically scale the number of instances running your bot, which allows your bot to keep up with demand. For more information about autoscaling, see [Autoscaling best practices](#).

For other scalability topics, see the Azure Architecture Center [Performance efficiency checklist](#).

DevOps

It's a common practice to deploy web apps, API apps, and mobile apps to an Azure App Service plan by using continuous deployment pipelines. Because a secured bot's app service is protected with a private endpoint, externally hosted build agents don't have the access that's required to deploy updates. To work around this, you might need to use a solution such as Azure Pipeline [self-hosted DevOps agents](#).

Security

Azure DDoS Protection, combined with application-design best practices, provides enhanced DDoS mitigation features to provide more defense against DDoS attacks. You should enable [Azure DDOS Protection](#) on any perimeter virtual network.

Deploy this scenario

Prerequisites

You must have an existing Azure account. If you don't have an Azure subscription, create a [free account](#) before you begin.

Walkthrough

1. Run the following Azure CLI commands in Azure Cloud Shell or your preferred deployment shell.

This set of commands creates the necessary resource group, virtual network, and subnets that are required for this walkthrough. The IP range used by Teams is [52.112.0.0/14,52.122.0.0/15](#).

```

# Declare variables (bash syntax)
export PREFIX='SecureBot'
export RG_NAME='rg-${PREFIX}'
export VNET_NAME='vnet-${PREFIX}'
export SUBNET_INT_NAME='VnetIntegrationSubnet'
export SUBNET_PVT_NAME='PrivateEndpointSubnet'
export LOCATION='eastus'
export TEAMS_IP_RANGE='52.112.0.0/14 52.122.0.0/15'
export FIREWALL_NAME='afw-${LOCATION}-${PREFIX}'

# Create a resource group
az group create --name ${RG_NAME} --location ${LOCATION}

# Create a virtual network with a subnet for the firewall
az network vnet create \
--name ${VNET_NAME} \
--resource-group ${RG_NAME} \
--location ${LOCATION} \
--address-prefix 10.0.0.0/16 \
--subnet-name AzureFirewallSubnet \
--subnet-prefix 10.0.1.0/26

# Add a subnet for the Virtual network integration
az network vnet subnet create \
--name ${SUBNET_INT_NAME} \
--resource-group ${RG_NAME} \
--vnet-name ${VNET_NAME} \
--address-prefix 10.0.2.0/24

# Add a subnet where the private endpoint will be deployed for the app
service
az network vnet subnet create \
--name ${SUBNET_PVT_NAME} \
--resource-group ${RG_NAME} \
--vnet-name ${VNET_NAME} \
--address-prefix 10.0.3.0/24

```

When you create a private endpoint subnet, the private endpoint policies are disabled by default.

When the deployment is complete, you should see the following subnets within your virtual network:

Name	IPv4
AzureFirewallSubnet	10.0.1.0/26 (59 available)
VnetIntegrationSubnet	10.0.2.0/24 (251 available)
PrivateLinkSubnet	10.0.3.0/24 (251 available)

2. Deploy an Azure Firewall instance into the firewall subnet that you created in step 1 by running the following CLI commands:

```
Azure CLI

# Create a firewall
az network firewall create \
    --name ${FIREWALL_NAME} \
    --resource-group ${RG_NAME} \
    --location ${LOCATION}

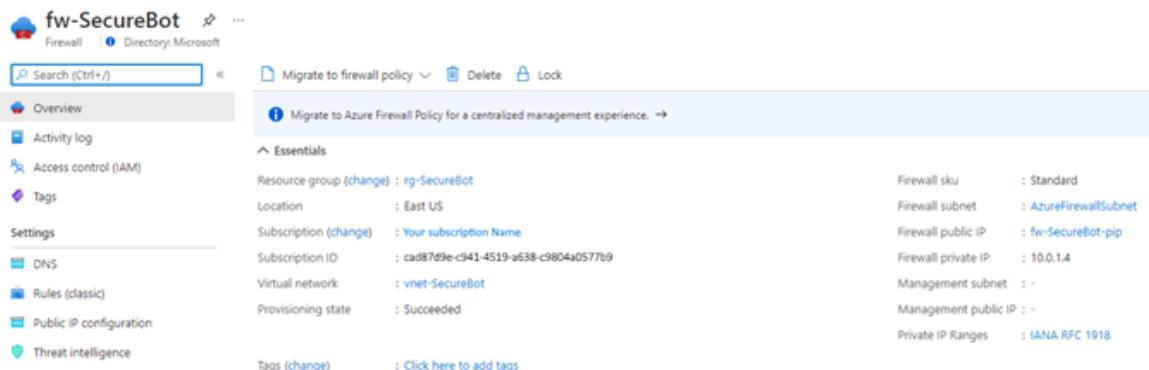
# Create a public IP for the firewall
az network public-ip create \
    --name ${FIREWALL_NAME}-pip \
    --resource-group ${RG_NAME} \
    --location ${LOCATION} \
    --allocation-method static \
    --sku standard

# Associate the IP with the firewall
az network firewall ip-config create \
    --firewall-name ${FIREWALL_NAME} \
    --name ${FIREWALL_NAME}-Config \
    --public-ip-address ${FIREWALL_NAME}-pip \
    --resource-group ${RG_NAME} \
    --vnet-name ${VNET_NAME}

# Update the firewall
az network firewall update \
    --name ${FIREWALL_NAME} \
    --resource-group ${RG_NAME}

# Get the public IP address for the firewall and take note of it for
# later use
az network public-ip show \
    --name ${FIREWALL_NAME}-pip \
    --resource-group ${RG_NAME}
```

Your firewall configuration should look something like this:



3. Create a basic bot.

4. Deploy the basic bot into the resource group that you created in step 1.

As part of this process, you'll create an app registration, which you need to interact with the bot via channels. During this process, you'll also deploy the necessary App Service plan, app service, and web app bot.

ⓘ Note

Select an App Service plan that supports Azure Private Link.

5. Map a custom domain to the app service that you deployed to the resource group in step 3.

This step requires access to your domain registrar, and it requires you to add an A-record to the custom domain that points to the public IP of the firewall you created in step 2.

6. Secure the mapped custom domain by either uploading an existing certificate for the domain or purchasing an App Service Certificate in Azure and importing it. You can do this by following the steps in [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

You should now have a fully functional bot that you can add to a channel in Teams or test through Web Chat by using the directions found in the [Bot Framework SDK documentation](#).

ⓘ Note

At this point the bot's app service is still publicly accessible over both the `azurewebsites.net` URL and over the custom URL you configured. In the next steps, you'll use private endpoints to disable public access. You'll also configure the firewall to allow the bot service to communicate only with Teams clients.

7. Run the following Azure CLI script to [deploy and configure the private endpoint](#).

This step also implements virtual network integration for the bot's app service, which connects it to your virtual network's integration subnet.

Azure CLI

```
# Disable private endpoint network policies (this step is not required if you're using the Azure portal)
az network vnet subnet update \
```

```

--name ${SUBNET_PVT_NAME} \
--resource-group ${RG_NAME} \
--vnet-name ${VNET_NAME} \
--disable-private-endpoint-network-policies true

# Create the private endpoint, being sure to copy the correct resource
# ID from your deployment of the bot app service
# The ID can be viewed by using the following CLI command:
# az resource show --name wapp-securebot --resource-group rg-securebot
# --resource-type Microsoft.web/sites --query "id"
az network private-endpoint create \
--name pvt-${PREFIX}Endpoint \
--resource-group ${RG_NAME} \
--location ${LOCATION} \
--vnet-name ${VNET_NAME} \
--subnet ${SUBNET_PVT_NAME} \
--connection-name conn-${PREFIX} \
--private-connection-resource-id /subscriptions/cad87d9e-c941-4519-
a638-c9804a0577b9/resourceGroups/rg-
securebot/providers/Microsoft.Web/sites/wapp-securebot \
--group-id sites

# Create a private DNS zone to resolve the name of the app service
az network private-dns zone create \
--name ${PREFIX}privatelink.azurewebsites.net \
--resource-group ${RG_NAME}

az network private-dns link vnet create \
--name ${PREFIX}-DNSLink \
--resource-group ${RG_NAME} \
--registration-enabled false \
--virtual-network ${VNET_NAME} \
--zone-name ${PREFIX}privatelink.azurewebsites.net

az network private-endpoint dns-zone-group create \
--name chatBotZoneGroup \
--resource-group ${RG_NAME} \
--endpoint-name pvt-${PREFIX}Endpoint \
--private-dns-zone ${PREFIX}privatelink.azurewebsites.net \
--zone-name ${PREFIX}privatelink.azurewebsites.net

# Establish virtual network integration for outbound traffic
az webapp vnet-integration add \
-g ${RG_NAME} \
-n wapp-${PREFIX} \
--vnet ${VNET_NAME} \
--subnet ${SUBNET_INT_NAME}

```

After you've run these commands, you should see the following resources in your resource group:

Name ↑↓	Type ↑↓	Location ↑↓
asp-wapp-SecureBot	App Service plan	East US
fw-SecureBot	Firewall	East US
fw-SecureBot-pip	Public IP address	East US
pvt-SecureBotEndpoint	Private endpoint	East US
pvt-SecureBotEndpoint.nic.f66d6885-13f6-4db9-84c1-d268723fb49c	Network interface	East US
securebotprivatelink.azurewebsites.net	Private DNS zone	Global
vnet-SecureBot	Virtual network	East US
wapp-securebot	App Service	East US

The VNet Integration option under the Networking section of your app service should look like this:

wapp-securebot | Networking

VNet Integration

Securely access resources available in or through your Azure VNet.

Click here to configure

Private Endpoint connections

Private access to services hosted on the Azure platform, keeping your data on th

Configure your private endpoint connections

Hybrid connections

Securely access applications in private networks

Configure your hybrid connection endpoints

Azure Front Door with Web Application Firewall

Scalable and secure entry point for accelerated delivery of your web applications

Configure Azure Front Door with WAF for your app

Azure CDN

Secure, reliable content delivery with broad global reach and rich feature set

Configure Azure CDN for your app

Access Restrictions

Home > wapp-securebot >

VNet Integration

wapp-securebot

[Disconnect](#) [Refresh](#)

VNet Configuration

Securely access resources available in or through your Azure VNet. [Learn more](#)

VNet Details

VNet NAME	vnet-SecureBot
LOCATION	East US

VNet Address Space

Start Address	End Address
10.0.0.0	10.0.255.255

Subnet Details

Subnet NAME	VnetIntegrationSubnet
-------------	-----------------------

Subnet Address Space

Start Address	End Address
10.0.2.0	10.0.2.255

Home > wapp-securebot >

Private Endpoint connections

[Add](#) [Refresh](#) | [Approve](#) [Reject](#) [Remove](#)

Private Endpoint connections

Private access to services hosted on the Azure platform, keeping your data on the Microsoft network. [Learn more](#)

Filter by name or description	All connection states	Connection name ↑↓	Connection state ↑↓	Private endpoint ↑↓
conn-SecureBot-8c1b4502-afaf-424d-8744-cfa0469cfabb	Approved	conn-SecureBot-8c1b4502-afaf-424d-8744-cfa0469cfabb	Approved	pvt-SecureBotEndpoint

8. Next, you create a route table to ensure that traffic to and from each subnet goes through the firewall. You'll need the private IP address of the firewall that you created in the previous step.

Azure CLI

```
# Create a route table
az network route-table create \
-g ${RG_NAME} \
-n rt-${PREFIX}RouteTable

# Create a default route with 0.0.0.0/0 prefix and the next hop as the
# Azure firewall virtual appliance to inspect all traffic. Make sure you
# use your firewall's internal IP address instead of 10.0.1.4
az network route-table route create -g ${RG_NAME} \
--route-table-name rt-${PREFIX}RouteTable -n default \
--next-hop-type VirtualAppliance \
--address-prefix 0.0.0.0/0 \
--next-hop-ip-address 10.0.1.4
```

```

# Associate the two subnets with the route table
az network vnet subnet update -g ${RG_NAME} \
-h ${SUBNET_INT_NAME} --vnet-name ${VNET_NAME} \
--route-table rt-${PREFIX}RouteTable

az network vnet subnet update -g ${RG_NAME} \
-h ${SUBNET_PVT_NAME} \
--vnet-name ${VNET_NAME} \
--route-table rt-${PREFIX}RouteTable

```

After you've run the commands, your route table resource should look like this:

Name	Address prefix	Next hop type
default	0.0.0.0	10.0.1.4

Name	Address range	Virtual network	Security group
VnetIntegrationSubnet	10.0.2.0/24	vnet-SecureBot	-
PrivateLinkSubnet	10.0.3.0/24	vnet-SecureBot	-

After you've created the route table, you add rules to your firewall to deliver traffic from the public IP to the bot app service, and to restrict traffic from any endpoint other than Microsoft Teams. In addition, you'll allow traffic between the virtual network and Azure Bot Services or Microsoft Entra ID by using service tags.

9. Run the following commands:

```

Azure CLI

# Create a NAT rule collection and a single rule. The source address is
# the public IP range of Microsoft Teams
# Destination address is that of the firewall.
# The translated address is that of the app service's private link.
az network firewall nat-rule create \
--resource-group ${RG_NAME} \
--collection-name coll-${PREFIX}-nat-rules \
--priority 200 \
--action DNAT \
--source-addresses ${TEAMS_IP_RANGE} \
--dest-addr 23.100.26.84 \
--destination-ports 443 \
--firewall-name ${FIREWALL_NAME} \
--name rl-ip2appservice \
--protocols TCP \
--translated-address 10.0.3.4 \

```

```

--translated-port 443

# Create a network rule collection and add three rules to it.
# The first one is an outbound network rule to only allow traffic to
the Teams IP range.
# The source address is that of the virtual network address space,
destination is the Teams IP range.
az network firewall network-rule create \
    --resource-group ${RG_NAME} \
    --collection-name coll-${PREFIX}-network-rules \
    --priority 200 \
    --action Allow \
    --source-addresses 10.0.0.0/16 \
    --dest-addr ${TEAMS_IP_RANGE} \
    --destination-ports 443 \
    --firewall-name ${FIREWALL_NAME} \
    --name rl-OutboundTeamsTraffic \
    --protocols TCP

# This rule will enable traffic to all IP addresses associated with
Azure AD service tag
az network firewall network-rule create \
    --resource-group ${RG_NAME} \
    --collection-name coll-${PREFIX}-network-rules \
    --source-addresses 10.0.0.0/16 \
    --dest-addr AzureActiveDirectory \
    --destination-ports '*' \
    --firewall-name ${FIREWALL_NAME} \
    --name rl-AzureAD \
    --protocols TCP

# This rule will enable traffic to all IP addresses associated with
Azure Bot Services service tag
az network firewall network-rule create \
    --resource-group ${RG_NAME} \
    --collection-name coll-${PREFIX}-network-rules \
    --source-addresses 10.0.0.0/16 \
    --dest-addr AzureBotService \
    --destination-ports '*' \
    --firewall-name ${FIREWALL_NAME} \
    --name rl-AzureBotService \
    --protocols TCP

```

After you've run the commands, your firewall rules will look something like this:

Name	Priority	Action	Source	Destination Addresses	Destination Ports	Translated address	Translated port
rl-ip2appservice	200	Destination Network Address Translation (DNAT)	52.112.0.0/14	23.100.26.84	443	10.0.3.4	443

The screenshot shows the 'Edit network rule collection' interface in the Azure portal. It displays three main sections: 'IP Addresses', 'Service Tags', and 'FQDNs'. Each section contains a table with columns for 'name', 'Protocol', 'Source type', 'Source', 'Destination type', 'Destination Addresses', and 'Destination Ports'. The 'IP Addresses' section has one rule named 'rl-OutboundTeamsTraffic' with TCP protocol, source type 'IP address' (10.0.0.0/16), source '*, 192.168.10.1, 192.168.10.0/24, ...', destination type 'IP address' (52.112.0.0/14), destination '*, 192.168.10.1, 192.168.10.0/24, ...', and destination ports '8080, 8080-8090, *'. The 'Service Tags' section has two rules: 'rl-AzureAD' (TCP, 10.0.0.0/16, AzureActiveDirectory) and 'rl-AzureBotService' (TCP, 10.0.0.0/16, AzureBotService). The 'FQDNs' section has one rule with source type 'IP address' (10.0.0.0/16, *, 192.168.10.1, 192.168.10.0/24, 192.16...), destination FQDN 'time.windows.com', and destination ports '8080, 8080-8090, *'.

10. Confirm that your bot is accessible only from a channel in Teams, and that all traffic to and from the bot app service goes through your firewall.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Ali Jafry](#) | Cloud Solution Architect

Next steps

- Review the [Bot Framework SDK Documentation](#) to start building bots.
- See [Bots Secured Behind a Firewall & Teams](#).

Related resources

- Visit the [Azure Architecture Center](#) to review related architectures and guides.
- [Azure Firewall Architecture Guide - Azure Architecture Center](#)
- [Microsoft Entra IDaaS in Security Operations - Azure Example Scenarios](#)
- [Threat indicators for cyber threat intelligence in Microsoft Sentinel - Azure Example Scenarios](#)
- [Confidential computing on a healthcare platform - Azure Example Scenarios](#)
- [Hub-spoke network topology in Azure](#)

Azure and Dynamics 365 scenarios

Article • 06/14/2023

Dynamics 365  is a suite of modern commerce solutions that work together to connect your entire business. Use any or all of these Dynamics 365 applications, according to your needs:

- **Customer data platform**
 - [Customer data platform](#) . Increase your knowledge of your customers with a real-time customer data platform.
 - [Customer Insights](#) . Use AI and analytics to improve your customers' experiences.
 - [Customer Voice](#) . Collect, analyze, and track real-time feedback in a scalable feedback management solution.
- **Sales**
 - [Sales](#) . Connect sales teams to customers and implement individualized selling.
 - [Microsoft Relationship Sales](#) . Transform relationship selling by using Dynamics 365 Sales and LinkedIn Sales Navigator.
- **Service**
 - [Customer Service](#) . Provide self-service support, tailor customer engagements, elevate agent effectiveness, and more.
 - [Field Service](#) . Move from reactive to proactive to predictive service by using data insights.
 - [Remote Assist](#) . Solve problems in real time, bring critical information into view, and walk through a site without being on location.
- **Marketing** . Engage customers in real time, gain customers and earn loyalty, and personalize customer experiences by using AI.
- **Commerce**
 - [Commerce](#) . Deliver unified and personalized buying experiences for customers and partners.
 - [Connected Spaces](#) . Generate actionable insights from your space by using pre-built AI-powered skills.
 - [Fraud Protection](#) . Implement adaptive AI that continuously learns in order to protect you against payment fraud, bots, account takeover, and returns and discounts fraud.
- **Supply chain**
 - [Supply Chain Management](#) . Ensure business continuity by using agile distribution and manufacturing processes in the cloud and at the edge.
 - [Supply Chain Insights](#) . Make better supply chain decisions by using proactive risk mitigation via AI-powered insights.

- [Guides](#). Create step-by-step holographic instructions.
- [Intelligent Order Management](#). Meet your digital commerce needs and scale easily while supporting the latest fulfillment methods.
- **Small and medium business**
 - [Business Central](#). Connect operations across your small or medium-sized business.
 - [Customer Service](#). Streamline support by using a solution that simplifies processes and improves customer experiences.
- [Human Resources](#). Create a workplace where people and business thrive.
- [Finance](#). Maximize financial visibility and profitability.
- [Project Operations](#). Connect your project-centric business, from prospects to payments to profits, in one application.

This article provides summaries of solutions and architectures that use Dynamics 365 together with Azure services.

Watch this short video to learn how Dynamics 365 can help you streamline business operations, enhance inventory management, optimize order fulfillment, and more:

https://www.youtube-nocookie.com/embed/HgX-GDWgs_I

Solutions across Azure and Dynamics 365

Dynamics 365 CRM and ERP (general)

Architecture	Summary	Technology focus
Architectural approaches for the deployment and configuration of multitenant solutions	Learn about deploying and configuring a multitenant solution. Use Dynamics 365 to trigger an onboarding process when a sale is made to a new customer.	Multitenancy
Azure IoT reference architecture	Review a recommended architecture for IoT applications on Azure. IoT devices provide insights to Dynamics 365.	IoT
Create smart places by using Azure Digital Twins	Use Azure Digital Twins to create models of smart places from IoT device data. You can use this data to inform Dynamics 365 applications.	IoT

Architecture	Summary	Technology focus
Enterprise bot for employee productivity	Use Azure Bot Service and Azure Cognitive Services to build enterprise bots for internal productivity. Use data from Microsoft 365 calendars to access customer information in Dynamics 365.	AI
Enterprise-scale disaster recovery	Review a large-enterprise architecture for SharePoint, Dynamics CRM, and Linux web servers that runs on an on-premises datacenter and fails over to Azure infrastructure.	Management
Eventual consistency between multiple Power Apps instances	Handle dependent data in a resilient way in Power Apps. Includes information about replicating data between Dynamics 365 instances.	Web
Migrate master data services to Azure with CluedIn and Azure Purview	Use CluedIn and Azure Purview to migrate your master data services solution to Azure. Dynamics 365 users can natively use data from CluedIn without any extra setup or integration.	Databases
Multitenancy and identity management	Learn authentication, authorization, and identity management best practices for multitenant applications. In these architectures, Dynamics CRM tenants store user profiles in Azure AD.	Identity
SAP on Azure architecture design	Review a set of guiding tenets that you can use to help ensure the quality of SAP workloads that run on Azure. A recommended identity management system, Azure Active Directory, integrates with Dynamics CRM Online.	SAP
Serverless computing solution for LOB apps	Build and run customer onboarding applications without managing or maintaining infrastructure. Customer information is stored in Dynamics 365.	Serverless

Dynamics 365 Customer Engagement

Architecture	Summary	Technology focus
Clinical insights with Microsoft Cloud for Healthcare	Learn about an architecture that you can use to gather insights from clinical and medical data by using Microsoft Cloud for Healthcare. Dynamics 365 components include an application that stores a list of patients together with scores that indicate the urgency of their cases.	Web

Architecture	Summary	Technology focus
Virtual health on Microsoft Cloud for Healthcare	Learn about an architecture that you can use to develop a virtual health solution by using Microsoft Cloud for Healthcare. One component of the solution is an application deployed via Dynamics 365 that presents an aggregated view of patient data in Teams.	Web

Dynamics 365 Customer Insights

Architecture	Summary	Technology focus
Customer 360 with Azure Synapse and Dynamics 365 Customer Insights	Build an end-to-end Customer 360 solution by using Azure Synapse Analytics and Customer Insights.	Analytics
Enhanced customer dimension with Dynamics 365 Customer Insights	Use Customer Insights to create an enhanced customer dataset and make it available in Azure Data Lake for consumption by Azure Synapse Analytics.	Analytics

Dynamics 365 Sales

Architecture	Summary	Technology focus
Clinical insights with Microsoft Cloud for Healthcare	Learn about an architecture that you can use to gather insights from clinical and medical data by using Microsoft Cloud for Healthcare. Dynamics 365 components include an application that stores a list of patients together with scores that indicate the urgency of their cases.	Web

Service

Architecture	Summary	Technology focus
Scalable cloud applications and SRE	Build scalable cloud applications by using performance modeling and other principles and practices of site reliability engineering (SRE). Dynamics 365 is used for product catalog and customer service management.	Web

Small and medium business

Architecture	Summary	Technology focus
Modern data warehouse for small and medium business	Use Azure Synapse Analytics, Azure SQL Database, and Data Lake Storage to modernize SMB legacy and on-premises data. These solutions integrate easily with Dynamics 365.	Analytics

Related resources

- [Browse all Dynamics 365 architectures](#)

Customer 360 with Azure Synapse and Dynamics 365 Customer Insights

Customer Insights - Data

Azure Synapse Analytics

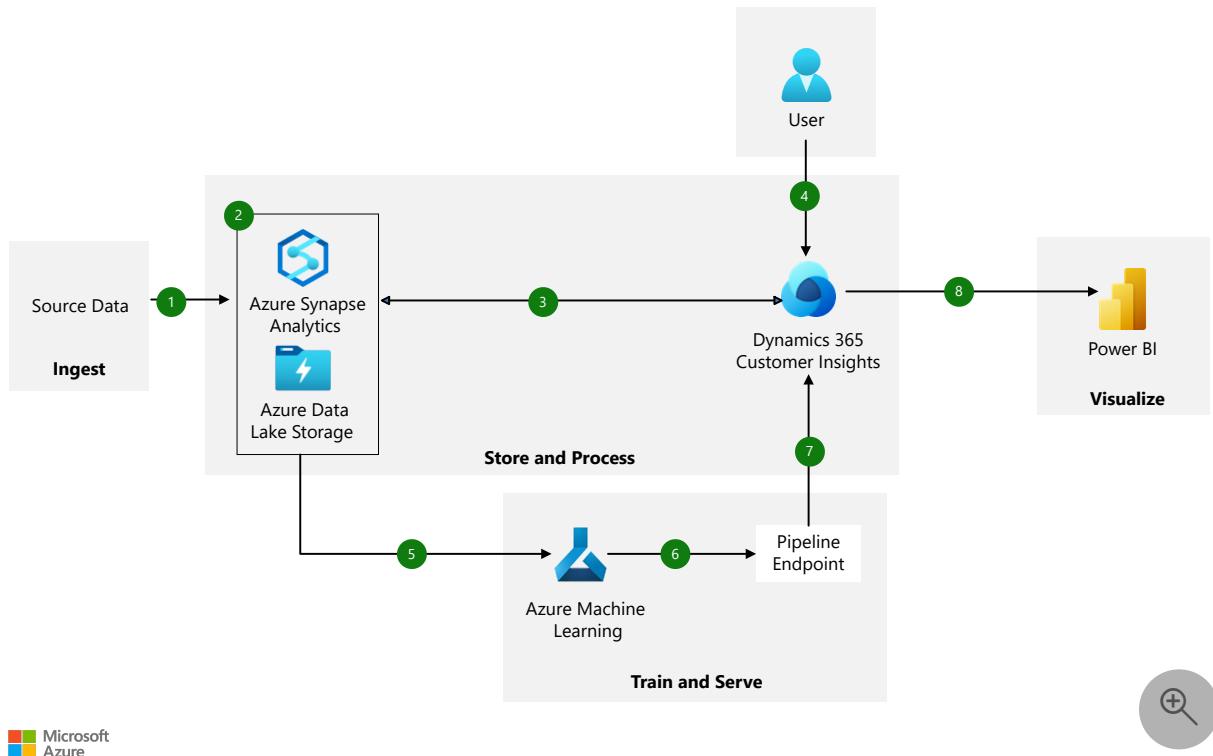
Azure Machine Learning

Power BI

This solution combines Azure Synapse Analytics with Dynamics 365 Customer Insights, to build a comprehensive view that presents your customer data and to provide the best customer experience.

Apache®, Apache Ignite, Ignite, and the flame logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of these marks.

Architecture



Download a [Visio file](#) of this architecture.

Dataflow

1. Azure Synapse Analytics ingests raw source data by using Azure Synapse pipelines.
2. Source data is stored in Azure Synapse Analytics and Azure Data Lake Storage Gen2.
3. Dynamics 365 Customer Insights connects to customer data from Azure Synapse.
4. Administrators configure unified customer profiles in Customer Insights, together with measures, segments, and enrichments. Unified customer profiles are ported from Customer Insights to Azure Synapse.
5. Administrators use the unified customer profile in Azure Synapse to create an Azure Machine Learning pipeline for retention prediction.
6. Administrators create a retention prediction model endpoint.
7. Administrators create a custom model workflow in Customer Insights to call the Azure Machine Learning pipeline to get the retention predictions.
8. Power BI ingests the Customer 360 data from Customer Insights to visualize the profiles and metrics.

Components

- [Dynamics 365 Customer Insights](#) can help you provide unmatched customer experiences by using world-class AI and analytics. Here, it's used to unify, segment, and enrich customer data.
- [Azure Synapse Analytics](#) is an analytics service that brings together data integration, enterprise data warehousing, and big data analytics. It's used here for data ingestion, storage, and processing.
- [Data Lake Storage](#) provides a massively scalable and secure data lake for your high-performance analytics workloads.
- [Azure Machine Learning](#) is an end-to-end machine learning service. It's used in this architecture to predict customer retention.
- [Power BI](#) can help you turn your data into coherent, visually immersive, and interactive insights. It's used here to visualize customer profiles and metrics.

Scenario details

Managing customer data from multiple sources and building a unified Customer 360 view isn't a new challenge. But it is becoming increasingly difficult with the increased number of interaction channels and touchpoints with customers. By combining Azure Synapse Analytics with Dynamics 365 Customer Insights, you can build a comprehensive view of your customers to provide the best customer experience.

Potential use cases

This solution was created for a property management organization. It can also be applied in industries like retail, financial services, manufacturing, and health care. It can be used by any organization that needs to bring data together across systems to build a Customer 360 profile and improve the customer experience.

You can use this solution to:

- Gain better insights from your customer data.
- Target sources of customer churn or dissatisfaction.
- Direct account and customer service activities.
- Run targeted promotions that are aimed at customer retention or upselling.

Considerations

These considerations implement the pillars of the Azure Well-Architected Framework, which is a set of guiding tenets that can be used to improve the quality of a workload. For more information, see [Microsoft Azure Well-Architected Framework](#).

Security

Security provides assurances against deliberate attacks and the abuse of your valuable data and systems. For more information, see [Overview of the security pillar](#).

This solution uses Microsoft Entra ID to authenticate users to the Azure solutions in the architecture. You can manage permissions via Microsoft Entra authentication or role-based access control.

Follow these security guidelines when you implement this solution:

- [Security in Azure](#)
- [Access control for Azure Synapse](#)
- [User permissions for Customer Insights](#)

Scalability

This solution uses Azure Synapse Spark clusters, which can be automatically scaled up and down based on the activity needs of your workload. For more information, see [Azure Synapse Spark cluster autoscaling](#).

Azure Machine Learning training pipelines can be scaled up and down based on data size and other configuration parameters. The compute clusters support autoscaling and automatic shutdown to optimize for performance and cost.

Cost optimization

Cost optimization is about looking at ways to reduce unnecessary expenses and improve operational efficiencies. For more information, see [Overview of the cost optimization pillar](#).

[Dynamics 365 Customer Insights](#) license pricing options are based on the number of customer profiles needed.

[Azure Synapse Analytics](#) has various pricing options to help you optimize costs. You can perform big data processing tasks like data engineering, data preparation, and machine learning directly in Azure Synapse by using memory-optimized or hardware-accelerated Apache Spark pools. Billing for usage of Spark pools is rounded up to the nearest minute.

[Azure Machine Learning](#) has no additional license charge. However, there are charges for compute and other Azure services that you consume, including but not limited to Azure Blob Storage, Azure Key Vault, Azure Container Registry, and Application Insights.

There are various [Power BI](#) product options to meet different requirements. [Power BI Embedded](#) provides an Azure-based option for embedding Power BI functionality in your applications.

You can deploy this solution with the following options.

- Dynamics 365 Customer Insights: 1,500 profiles
- Azure Synapse Analytics: Memory-optimized Spark cluster of medium size (8 vCores / 64 GB)
- Azure Machine Learning
 - Compute instance of type Standard_DS11_v2
 - Compute cluster of type Standard_D2_v2

Azure services like Azure Storage accounts, Key Vault, Container Registry, Application Insights, and so on, that are deployed with Azure Synapse Analytics and Azure Machine Learning incur other costs.

Deploy this scenario

To deploy this solution, follow the steps in the [Getting Started guide](#) and the step-by-step [Deployment Guide](#). You can find them in the [GitHub repository](#) for the solution.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Nalini Chandhi](#) | Sr. Technical Specialist

Next steps

- [Unlock customer intent with Dynamics 365 Customer Insights](#)
- [Product overview for Dynamics 365 Customer Insights](#)
- [What is Azure Machine Learning?](#)
- [What is Azure Synapse Analytics?](#)

Related resources

- [Enhanced customer dimension with Dynamics 365 Customer Insights](#)
- [Modern data warehouse for small and medium businesses](#)
- [Clinical insights with Microsoft Cloud for Healthcare](#)
- [Analytics architecture design](#)

Enhanced customer dimension with Dynamics 365 Customer Insights

Azure Data Lake Storage

Azure Synapse Analytics

Azure Data Factory

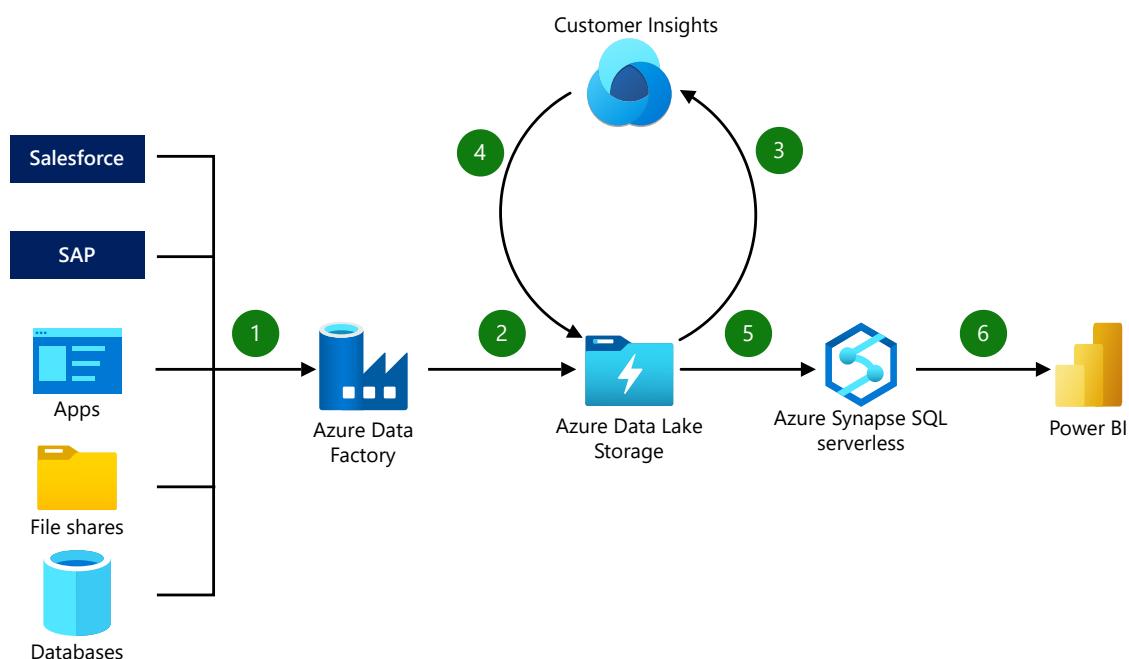
Customer Insights - Data

💡 Solution ideas

This article is a solution idea. If you'd like us to expand the content with more information, such as potential use cases, alternative services, implementation considerations, or pricing guidance, let us know by providing [GitHub feedback](#).

This high-level architecture shows the flow of data from an organization's source systems (ERP, CRM, POS, and so on) into a data lake on Azure. This same data lake can be configured as the back end for Dynamics 365 Customer Insights. When it has a data lake back end, Customer Insights can load clean enhanced customer data into the data lake for consumption as a dimension by downstream data warehouses and apps.

Architecture



[Download a Visio file](#) of this architecture.

Azure Synapse serverless SQL consumes the enhanced Customer Insights data. Azure Synapse serverless SQL introduces a cost-effective design pattern known as Logical Data Warehouse (LDW). The LDW pattern introduces an abstraction layer on top of external data stores, like data lakes, to provide familiar relational database constructs like tables and views. Tools that support SQL Server endpoints can then consume these tables and views. In the context of this example, Power BI can source the enhanced Customer Insights data as a dimension table from a database by using Azure Synapse serverless SQL pools.

Dataflow

1. By using Data Factory or Azure Synapse pipelines, establish [linked services](#) to source systems and data stores. Data Factory and Azure Synapse pipelines support [more than 90 connectors](#), including generic protocols for data sources when a native connector isn't available.
2. Load data from the source systems into Data Lake by using the [Copy Data tool](#). You then need to transform data in the data lake to fit a Common Data Model schema. Data Factory mapping data flows support sinking data in the Common Data Model format. For more information, see [Common Data Model format in Azure Data Factory and Synapse Analytics](#).
3. To import data into Customer Insights, you need to configure a [connection to a Common Data Model folder by using a Data Lake account](#). After you import data into Customer Insights, the Customer Insights [data unification process \(map, match, and merge\)](#) can process the disparate customer data. You can then further enrich unified data in Customer Insights by using [data enrichment](#), [data segments](#), and [AI predictions](#).
4. In Customer Insights, you need to configure an export of data back to the data lake. For more information, see [Set up the connection to Azure Data Lake Storage Gen2](#).
5. [Create a Logical Data Warehouse](#) in the Azure Synapse workspace. See the [Azure Synapse serverless SQL pool best practices](#) to determine whether you need to do more transformations on the exported Customer Insights data and whether views are better suited than tables.
6. Customer Insights data in the data lake is now exposed as logical SQL Server tables and views that can easily be consumed by Power BI. See [Tutorial for using serverless SQL pools with Power BI](#) for an example.

Components

- [Azure Data Lake Storage](#). Scalable and cost-effective cloud storage that Customer Insights supports as a target for exporting data.
- [Azure Data Factory](#). Cloud-scale data integration service for orchestrating data flow.
- [Audience insights](#). The Customer Insights module that unifies customer data sources. It also provides enrichments like segmentation, customer total lifetime value (CTLV), and customer churn score.
- [Azure Synapse serverless SQL pools](#). Used to query customer data in a data lake via T-SQL and SQL Server endpoint.

Alternatives

This solution uses the Logical Data Warehouse (LDW) pattern to consume the enhanced data from Customer Insights. You can also use other data warehouse patterns.

Data Factory and Azure Synapse both provide data integration pipelines. See the [breakdown of feature parity](#) for a comparison.

Scenario details

[Dynamics 365 Customer Insights](#) can create a 360-degree customer view by unifying data from transactional, behavioral, and observational sources. You can then make this 360-degree customer view available in enterprise data lakes and/or data warehouses as an enhanced customer [dimension](#).

This article describes the dataflow, product integrations, and configurations that are available for building an enhanced customer dimension that can be consumed by analytics platforms external to Dynamics 365 and Customer Insights. [Audience insights](#) is the feature of Customer Insights that provides the ability to unify customer data sources and enhance customer profiles. For more information, see [the audience insights overview](#).

The following table shows an example of enhanced customer records that are produced by the Customer Insights data unification process. This process takes customer data from multiple source systems and cleans and merges it. Customer Insights can also enrich customer records with attributes like churn scores and brand affinities. Here are some fictional examples of this type of record:

CustomerId	Name	Address	City	State	PostalCode	Country	EmailId	DOB	RewardsPoints	ChurnScore
e7b8e460e37...	Gustie Spacie	199 Burning Wood Point	Baton Rouge	Louisiana	70820	United States	gspacie@fabrikam.com	1975-01-13	263	0.230083182454109
2b23ec54b29...	Broddie Kempston	40 Muir Terrace	Chicago	Illinois	60669	United States	bkempson@proseware.com	1943-01-28	188	0.702696681022644
0019a5d0605...	Lorrain Ruffles	33 Brown Drive	Santa Monica	California	90410	United States	lruffles@treyresearch.net	1988-10-25	178	0.802143514156342

CustomerId	Name	Industry	Brand	AffinityScore	AffinityLevel	AgeDemographicSegment
e7b8e460e37...	Gustie Spacie	Computers & Consumer Electronics	Adatum Cor	48	Medium	AGE35_49
e7b8e460e37...	Gustie Spacie	Food & Groceries	Best for you	49	Medium	AGE35_49
e7b8e460e37...	Gustie Spacie	Retailers & General Merchandise	Northwind	74	High	AGE35_49
e7b8e460e37...	Gustie Spacie	Retailers & General Merchandise	Tailspin Toy	86	VeryHigh	AGE35_49

Potential use cases

This architecture is applicable to any organization that needs to create records that draw data from multiple sources.

This solution is optimized for the retail industry.

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Jon Dobrzeniecki](#) | Cloud Solution Architect

Next steps

- [Microsoft Learn: Unlock customer intent with Dynamics 365 audience insights](#)
- [Tutorial: Explore and analyze data lakes with serverless SQL pool](#)
- [Tutorial: Create a Logical Data Warehouse with serverless SQL pool](#)
- [Get started with Azure Synapse Analytics](#)
- [Customer Insights overview](#)
- [Analyze data in a storage account](#)
- [Integrate activities by using pipelines](#)

Related resources

- [Get started with analytics architecture design](#)
- [Choose an analytical data store in Azure](#)
- [Analytics end-to-end with Azure Synapse](#)

Scenarios featuring Microsoft on-premises technologies on Azure

Article • 11/01/2023

This article describes scenarios that feature Microsoft on-premises technologies on Azure. These scenarios enable you to gain the benefits provided by the cloud when you use technologies that were historically used in a local environment. The following on-premises technologies are frequently used in Azure solutions:

- [Windows Server 2022](#). The foundation of the Microsoft ecosystem. It continues to power the hybrid cloud network.
- [SQL Server 2019](#). The data platform that's known for performance, security, and availability.
- [Exchange Server](#). The messaging platform that provides email, scheduling, and tools for custom collaboration and messaging service applications.
- [Active Directory Domain Services in Windows Server 2016](#). The service that stores information about user accounts and enables other authorized users on the same network to access it. Security is integrated with Active Directory through sign-in authentication and access control to objects in the directory.
- [Host Integration Server](#). Technologies and tools that enable enterprise organizations to integrate existing IBM host systems, programs, messages, and data with new Microsoft server applications.

For information about solutions in which Azure services integrate with the other Microsoft cloud platforms, see these articles:

- [Azure and Power Platform scenarios](#)
- [Azure and Microsoft 365 scenarios](#)
- [Azure and Dynamics 365 scenarios](#)

Active Directory

Architecture	Summary	Technology focus
Azure files secured by AD DS	Provide high-security file access with on-premises Windows Server Active Directory Domain Services (AD DS) and DNS.	Hybrid
Create an AD DS resource forest	Learn how to create a separate Active Directory domain on Azure that's trusted by domains in your	Identity

Architecture	Summary	Technology focus
	on-premises Active Directory forest.	
Deploy AD DS in an Azure virtual network	Use this reference architecture to extend an on-premises Active Directory domain to Azure to provide distributed authentication services.	Identity
Disaster recovery for Azure Stack Hub VMs	Learn about an optimized approach to disaster recovery of VM-based user workloads that are hosted on Azure Stack Hub. Azure Site Recovery integrates with Windows Server-based apps and roles, including Active Directory Domain Services.	Hybrid
Extend on-premises AD FS to Azure	Implement a highly secure hybrid network architecture by using Active Directory Federation Services (AD FS) authorization on Azure.	Identity
Hybrid SharePoint farm with Microsoft 365	Deliver highly available intranet capability and share hybrid workloads with Microsoft 365 by using SharePoint servers, Microsoft Entra ID, and SQL Server. Windows Server hosts Active Directory services for service and machine accounts.	Hybrid
Integrate on-premises Active Directory with Azure	Compare options for integrating your on-premises Active Directory environment with an Azure network.	Identity
Multiple forests with AD DS and Microsoft Entra ID	Create multiple Active Directory forests by using Azure Virtual Desktop.	Virtual Desktop
Multiple forests with AD DS, Microsoft Entra ID, and Microsoft Entra Domain Services	Create multiple Active Directory forests by using Azure Virtual Desktop and AD DS.	Virtual Desktop
On-premises Active Directory domains with Microsoft Entra ID	Learn how to implement a secure hybrid network architecture that integrates on-premises Active Directory domains with Microsoft Entra ID.	Identity
Use Azure file shares in a hybrid environment	Use identity-based authentication to control access to Azure file shares via AD DS users and groups.	Hybrid

Exchange Server

Architecture	Summary	Technology focus
Back up files and apps on Azure Stack Hub	Learn about an optimized approach to backing up and restoring files and applications of VM-based user workloads that are hosted on Azure Stack Hub. Includes backup and restore of Exchange Server servers and databases.	Hybrid
Disaster recovery for Azure Stack Hub VMs	Learn about an optimized approach to disaster recovery of VM-based user workloads that are hosted on Azure Stack Hub. Includes a discussion of disaster recovery for Exchange workloads.	Hybrid
Enhanced-security hybrid messaging - client access	Enhance your security in a client access scenario by using Microsoft Entra multifactor authentication. Discusses scenarios for Exchange Online and Exchange on-premises.	Hybrid
Enhanced-security hybrid messaging - mobile access	Enhance your security in a mobile access scenario by using Microsoft Entra multifactor authentication. Discusses scenarios for Exchange Online and Exchange on-premises.	Hybrid
Enhanced-security hybrid messaging - web access	Enhance your security in a web access scenario by using Microsoft Entra multifactor authentication. Discusses scenarios for Exchange Online and Exchange on-premises.	Hybrid

Host Integration Server (HIS)

Architecture	Summary	Technology focus
Integrate IBM mainframe and midrange message queues with Azure	Learn about a data-first approach to middleware integration that enables IBM message queues. HIS is used in a VM-based IaaS approach.	Mainframe
Mainframe file replication and sync on Azure	Learn several options for moving, converting, transforming, and storing mainframe and midrange file system data on-premises and on Azure. HIS is used to convert EBCDIC files to make them compatible with Azure.	Mainframe

SharePoint Server

Architecture	Summary	Technology focus
Back up files and apps on Azure Stack Hub	Learn about an optimized approach to backing up and restoring files and applications of VM-based user workloads that are hosted on Azure Stack Hub. Includes backup and restore of SharePoint farms and front-end web server content and restore of SharePoint databases, web apps, files, list items, and search components.	Hybrid
Disaster recovery for Azure Stack Hub VMs	Learn about an optimized approach to disaster recovery of VM-based user workloads that are hosted on Azure Stack Hub. Includes information about disaster recovery for SharePoint workloads.	Hybrid
Highly available SharePoint farm	Deploy a highly available SharePoint farm for intranet capabilities that uses Microsoft Entra ID, a SQL Server Always On instance, and SharePoint resources.	Web
Hybrid SharePoint farm with Microsoft 365	Deliver highly available intranet capability and share hybrid workloads with Microsoft 365 by using SharePoint servers, Microsoft Entra ID, and SQL Server.	Hybrid
Multitier web application built for HA/DR	Learn how to create a resilient multitier web application that's built for high availability and disaster recovery on Azure. Applies to applications like SharePoint.	Networking
On-premises data gateway for Logic Apps	Review a reference architecture that illustrates a logic app that runs in Azure and connects to on-premises resources like SharePoint Server.	Hybrid
Run a highly available SharePoint Server 2016 farm in Azure	Learn proven practices for deploying a highly available SharePoint Server 2016 farm on Azure.	Management

SQL Server

Architecture	Summary	Technology focus
Back up files and apps on Azure Stack Hub	Learn about an optimized approach to backing up and restoring files and applications of VM-based user workloads that are hosted on Azure Stack Hub. Includes backup and restore of SQL Server instances and their databases.	Hybrid
Data integration with Logic Apps and SQL Server	Automate data integration tasks by using Azure Logic Apps. Configure API calls to trigger tasks	Integration

Architecture	Summary	Technology focus
Disaster recovery for Azure Stack Hub VMs	Learn about an optimized approach to disaster recovery of VM-based user workloads that are hosted on Azure Stack Hub. Includes information about disaster recovery for SQL Server workloads.	Hybrid
Enterprise business intelligence	Learn how to implement an ELT pipeline that moves data from an on-premises SQL Server database into Azure Synapse Analytics and transforms the data for analysis.	Integration
Hybrid SharePoint farm with Microsoft 365	Deliver highly available intranet capability and share hybrid workloads with Microsoft 365 by using SharePoint servers, Microsoft Entra ID, and SQL Server.	Hybrid
Loan credit risk and default modeling	Learn how SQL Server 2016 with R Services can help lenders issue fewer unprofitable loans by predicting borrower credit risk and default probability.	Databases
Mainframe access to Azure databases	Give mainframe applications access to Azure data without changing code. Use Microsoft Service for DRDA to run Db2 SQL statements on a SQL Server database.	Mainframe
Manage data across your Azure SQL estate with Azure Purview	Improve your organization's governance process by using Azure Purview in your Azure SQL estate.	Analytics
Micro Focus Enterprise Server on Azure VMs	Optimize, modernize, and streamline IBM z/OS mainframe applications by using Micro Focus Enterprise Server 6.0 on Azure VMs. This solution uses a SQL Server IaaS database in an Always On cluster.	Mainframe
On-premises data gateway for Logic Apps	Review a reference architecture that illustrates a logic app that runs in Azure and connects to on-premises resources like SQL Server.	Hybrid
Optimize administration of SQL Server instances in on-premises and multicloud environments by using Azure Arc	Learn how to use Azure Arc for the management, maintenance, and monitoring of SQL Server instances in on-premises and multicloud environments.	Databases
Oracle database migration: Rearchitect	Rearchitect your Oracle database by using Azure SQL Managed Instance.	Migration

Architecture	Summary	Technology focus
Plan deployment for updating Windows VMs on Azure	Learn best practices for configuring your environment for Windows Server Update Services (WSUS). SQL Server is the recommended database for servers that support a high number of client computers.	Management
Run a highly available SharePoint Server 2016 farm in Azure	Learn proven practices for deploying a highly available SharePoint Server 2016 farm on Azure by using MinRole topology and SQL Server Always On availability groups.	Management
SAP NetWeaver on SQL Server	Build an SAP landscape on NetWeaver by using Azure Virtual Machines to host SAP applications and a SQL Server database.	SAP
SQL Server 2008 R2 failover cluster on Azure	Learn how to rehost SQL Server 2008 R2 failover clusters on Azure virtual machines.	Databases
SQL Server on Azure Virtual Machines with Azure NetApp Files	Implement a high-bandwidth, low-latency solution for SQL Server workloads. Use Azure NetApp Files for enterprise-scale performance and reduced costs.	Storage
Web app private connectivity to Azure SQL Database	Lock down access to an Azure SQL database with Azure Private Link connectivity from a multitenant web app.	Web

Browse all our SQL Server solutions.

Windows Server

Architecture	Summary	Technology focus
Azure files secured by AD DS	Provide high-security file access with on-premises Windows Server AD DS and DNS.	Hybrid
Back up files and apps on Azure Stack Hub	Learn about an optimized approach to backing up and restoring files and applications of VM-based user workloads that are hosted on Azure Stack Hub. Supports backup and restore of various resources on VMs that run Windows Server.	Hybrid
Connect standalone servers by using	Learn how to connect an on-premises standalone server to Azure virtual networks by using Azure Network	Hybrid

Architecture	Summary	Technology focus
Azure Network Adapter	Adapter. Deploy Network Adapter by using Windows Admin Center on Windows Server.	
Disaster recovery for Azure Stack Hub VMs	Learn about an optimized approach to disaster recovery of VM-based user workloads that are hosted on Azure Stack Hub. Includes information about providing disaster recovery for Azure Stack Hub VMs that run Windows Server operating systems.	Hybrid
Manage hybrid Azure workloads using Windows Admin Center	Learn how to design a hybrid Windows Admin Center solution to manage workloads that are hosted on-premises and on Azure.	Hybrid
Plan deployment for updating Windows VMs on Azure	Learn best practices for configuring your environment for WSUS.	Management
Run a highly available SharePoint Server 2016 farm on Azure	Learn proven practices for deploying a highly available SharePoint Server farm on Azure. Windows Server Active Directory domain controllers run in the virtual network and have a trust relationship with the on-premises Windows Server Active Directory forest.	Management
Run SAP NetWeaver in Windows on Azure	Learn proven practices for running SAP NetWeaver in a Windows environment on Azure.	SAP
SQL Server 2008 R2 failover cluster on Azure	Learn how to rehost SQL Server 2008 R2 failover clusters on Azure virtual machines. Use the Azure shared disks feature and a Windows Server 2008 R2 failover cluster to replicate your on-premises deployment on Azure.	Databases

Related resources

- [Microsoft partner and third-party scenarios on Azure](#)
- [Architecture for startups](#)
- [Azure and Power Platform scenarios](#)
- [Azure and Microsoft 365 scenarios](#)
- [Azure and Dynamics 365 scenarios](#)
- [Azure for AWS professionals](#)
- [Azure for Google Cloud professionals](#)

Apache open-source scenarios on Azure

Article • 11/01/2023

Microsoft is proud to support open-source projects, initiatives, and foundations and contribute to thousands of open-source communities. By using open-source technologies on Azure, you can run applications your way while optimizing your investments.

This article provides a summary of architectures and solutions that use Azure together with Apache open-source solutions.

Apache®, Apache Ignite, Ignite, and the flame logo are either registered trademarks or trademarks of the Apache Software Foundation in the United States and/or other countries. No endorsement by The Apache Software Foundation is implied by the use of these marks.

Apache Cassandra

Architecture	Summary	Technology focus
Data partitioning guidance	View guidance for how to separate data partitions to be managed and accessed separately. Understand horizontal, vertical, and functional partitioning strategies. Cassandra is ideally suited to vertical partitioning.	Databases
High availability in Azure public MEC	Learn how to deploy workloads in active-standby mode to achieve high availability and disaster recovery in Azure public multi-access edge compute. Cassandra can be used to support geo-replication.	Hybrid
IoT and data analytics	Build solutions that integrate data from many IoT devices into a comprehensive data analysis architecture to improve and automate decision making. In this scenario, a Cassandra cluster is used to store data.	Analytics
N-tier application with Apache Cassandra	Deploy Linux virtual machines and a virtual network configured for an N-tier architecture with Apache Cassandra.	Databases
Non-relational data and NoSQL	Learn about non-relational databases that store data as key-value pairs, graphs, time series, objects, and other storage models, based on data requirements. Azure Cosmos DB for Apache Cassandra is a recommended Azure service.	Databases

Architecture	Summary	Technology focus
Run Apache Cassandra on Azure VMs	Examine performance considerations for running Apache Cassandra on Azure virtual machines. Use these recommendations as a baseline to test against your workload.	Databases
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use open-source technologies like Kafka, Kubernetes, Cassandra, PostgreSQL, and Redis components.	Analytics

Apache CouchDB

Architecture	Summary	Technology focus
Baseline web application with zone redundancy	Use the proven practices in this reference architecture to improve redundancy, scalability and performance in an Azure App Service web application. CouchDB is a recommended document database.	Web

Apache Hadoop

Architecture	Summary	Technology focus
Actuarial risk analysis and financial modeling	Learn how an actuarial developer can move an existing solution and its supporting infrastructure to Azure. Use Hadoop for data analysis.	Analytics
AI at the edge with Azure Stack Hub	Bring your trained AI model to the edge with Azure Stack Hub. Integrate it with your applications for low-latency intelligence. Use Hadoop to store data.	AI
AI at the edge with Azure Stack Hub - disconnected	Take advantage of edge AI when disconnected from the internet and move your AI models to the edge with a solution that includes Azure Stack Hub. Use Hadoop to store data.	AI
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. Azure HDInsight Hadoop clusters can be used for batch processing.	Databases

Architecture	Summary	Technology focus
Choose a data transfer technology	Learn about Azure data transfer options like Azure Import/Export service, Azure Data Box, Azure Data Factory, and command-line and graphical interface tools. The Hadoop ecosystem provides tools for data transfer.	Databases
Citizen AI with Power Platform	Learn how to use Azure Machine Learning and Power Platform to quickly create a machine learning proof of concept and production version. Azure Data Lake, a Hadoop-compatible file system, stores data.	AI
Data considerations for microservices	Learn about managing data in a microservices architecture. View an example that uses Azure Data Lake Store, a Hadoop file system.	Microservices
Extend your on-premises big data investments with HDInsight	Extend your on-premises big data investments to the cloud. Transform your business by using the advanced analytics capabilities of HDInsight. Hadoop is used as a data store.	Analytics
Extract actionable insights from IoT data	Extract insights from IoT data by using Azure services. HDInsight, a managed Hadoop service, can be used to process and transform data in cold storage.	Analytics
Extract, transform, and load	Learn about extract-transform-load (ETL) and extract-load-transform (ELT) data transformation pipelines and how to use control flows and data flows. Hadoop can be used as destination data store in ELT processes.	Analytics
ETL using HDInsight	ETL big data clusters on demand by using HDInsight, Hadoop MapReduce, and Apache Spark.	Analytics
IoT analyze-and-optimize loops	Learn about analyze-and-optimize loops, an IoT pattern for generating and applying optimization insights based on an entire business context. Hadoop map-reduce processing can be used to process big data.	IoT
Master data management with Azure and CluedIn	Use CluedIn eventual connectivity data integration to blend data from many siloed data sources and prepare it for analytics and business operations. CluedIn can take input data from Hadoop.	Databases
Materialized View pattern	Generate prepopulated views over the data in one or more data stores when the data isn't ideally formatted for your required query operations. Use Hadoop for a big data storage mechanism that supports indexing.	Databases

Architecture	Summary	Technology focus
Predict loan charge-offs with HDInsight Spark	Use HDInsight and machine learning to predict the likelihood of loans getting charged off. HDInsight supports Hadoop.	Databases
Predictive maintenance for industrial IoT	Connect devices that use the Open Platform Communications Unified Architecture standard to the cloud, and use predictive maintenance to optimize production. Use HDInsight with Hadoop to visually transform data.	IoT

Apache HBase

Architecture	Summary	Technology focus
AI at the edge with Azure Stack Hub	Bring your trained AI model to the edge with Azure Stack Hub. Integrate it with your applications for low-latency intelligence. Use HBase to store data.	AI
AI at the edge with Azure Stack Hub - disconnected	Take advantage of edge AI when disconnected from the internet and move your AI models to the edge with a solution that includes Azure Stack Hub. Use HBase to store data.	AI
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. You can use HBase for data presentation in these scenarios.	Databases
Choose a big data storage technology	Compare big data storage technology options in Azure. Includes a discussion of HBase on HDInsight.	Databases
Choose an analytical data store	Learn about using HBase for random access and strong consistency for large amounts of unstructured and semi-structured data.	Analytics
Data partitioning guidance	View guidance for separating data partitions so they can be managed and accessed separately. Understand horizontal, vertical, and functional partitioning strategies. HBase is ideally suited to vertical partitioning.	Databases
Non-relational data and NoSQL	Learn about non-relational databases that store data as key-value pairs, graphs, time series, objects, and other storage models, based on data requirements. HBase can be used for columnar and time series data.	Databases

Apache Hive

Architecture	Summary	Technology focus
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. You can use Hive for batch processing and data presentation in these scenarios.	Databases
Campaign optimization with HDInsight Spark	Build and deploy a machine learning model to maximize the purchase rate of leads that are targeted by a marketing campaign. Hive is used to store recommendations for how and when to contact each lead.	Databases
Choose a batch processing technology	Compare technology choices for big data batch processing in Azure. Learn about the capabilities of Hive.	Analytics
Choose an analytical data store	Evaluate analytical data store options for big data in Azure. Learn about the capabilities of Hive.	Analytics
Extract, transform, and load	Learn about ETL and ELT data transformation pipelines and how to use control flows and data flows. In ELT, you can use Hive to query source data. You can also use it together with Hadoop as a data store.	Databases
ETL using HDInsight	ETL big data clusters on demand by using HDInsight, Hive, and Apache Spark.	Analytics
Loan charge-off prediction with HDInsight Spark clusters	Use HDInsight and machine learning to predict the likelihood of loans getting charged off. Analytics results are stored in Hive tables.	Analytics
Predictive aircraft engine monitoring	Learn how to combine real-time aircraft data with analytics to create a solution for predictive aircraft engine monitoring and health. Hive scripts provide aggregations on raw events that are archived by Azure Stream Analytics.	Analytics
Predictive insights with vehicle telematics	Learn how car dealerships, manufacturers, and insurance companies can use Azure to get predictive insights on vehicle health and driving habits. In this solution, Azure Data Factory uses HDInsight to run Hive queries to process and load data.	Analytics

Architecture	Summary	Technology focus
Scale AI and machine learning initiatives in regulated industries	Learn about scaling Azure AI and machine learning environments that must comply with extensive security policies. Hive is used to store metadata.	AI

Apache JMeter

Architecture	Summary	Technology focus
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. A custom JMeter solution is used for load testing.	Migration
JMeter implementation for a load testing pipeline	Get an overview of an implementation for a scalable cloud load-testing pipeline.	Migration
Patterns and implementations for a banking cloud transformation	Learn about the patterns and implementations used to transform a banking system for the cloud. JMeter is used for load testing.	Migration
Scalable cloud applications and SRE	Build scalable cloud applications by using performance modeling and other principles and practices of site reliability engineering (SRE). JMeter is used for load testing.	Web
Unified logging for microservices apps	Learn about logging, tracing, and monitoring for microservices apps. JMeter is recommended for testing the behavior and performance of services.	Microservices

Apache Kafka

Architecture	Summary	Technology focus
Anomaly detector process	Learn about Anomaly Detector and see how anomaly detection models are selected with time series data. In this architecture, Event Hubs for Kafka can be used as an alternative to running your own Kafka cluster.	Analytics
Application data protection for AKS	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility	Containers

Architecture	Summary	Technology focus
workloads on Azure NetApp Files	for Azure Kubernetes Service (AKS) applications, including Kafka applications.	
Asynchronous messaging options	Learn about asynchronous messaging options in Azure, including support for Kafka clients.	Integration
Automated guided vehicles fleet control	Learn about an end-to-end approach for an automotive original equipment manufacturer (OEM). Includes several open-source libraries that you can reuse. Back-end services in this architecture can connect to Kafka.	Web
Azure Data Explorer interactive analytics	Ingest Kafka data into Azure Data Explorer and examine it by using improvised, interactive, fast queries.	Analytics
Azure Data Explorer monitoring	Use Azure Data Explorer in a hybrid monitoring solution that ingests streamed and batched logs from Kafka and other sources.	Analytics
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. Events from Event Hubs for Kafka feed into the system.	Containers
Choose a stream processing technology	Compare options for real-time message stream processing in Azure, including the Kafka streams API.	Analytics
Claim-Check pattern	Examine the Claim-Check pattern, which splits a large message into a claim check and a payload to avoid overwhelming a message bus. Learn about an example that uses Kafka for claim-check generation.	Integration
Data streaming with AKS	Use AKS to easily ingest and process a real-time data stream with millions of data points collected via sensors. Kafka stores data for analysis.	Containers
Extract actionable insights from IoT data	Extract insights from IoT data by using Azure services. Kafka on HDInsight is one option for ingesting the data stream.	Serverless
Ingestion, ETL, and stream processing pipelines with Azure Databricks	Create ETL pipelines for batch and streaming data with Azure Databricks to simplify data lake ingestion at any scale. Kafka is one option for ingesting data.	Analytics
Integrate Event Hubs with Azure Functions	Learn how to architect, develop, and deploy efficient and scalable code that runs on Azure Functions and	Serverless

Architecture	Summary	Technology focus
	responds to Azure Event Hubs events. Learn how events can be persisted in Kafka topics.	
IoT analytics with Azure Data Explorer	Use Azure Data Explorer for near real-time IoT telemetry analytics on fast-flowing, high-volume streaming data from a variety of data sources, including Kafka.	Analytics
JMeter implementation for a load testing pipeline	Get an overview of an implementation for a scalable cloud load-testing pipeline. The implementation supports reporting on the Kafka partitioning strategies used.	Migration
Mainframe and midrange data replication to Azure using Qlik	Use Qlik Replicate to migrate mainframe and midrange systems to the cloud, or to extend such systems with cloud applications. In this solution, Kafka stores change log information that's used to replicate the data stores.	Mainframe
Partitioning in Event Hubs and Kafka	Learn about partitioning in Kafka and Event Hubs for Kafka. Learn how many partitions to use in ingestion pipelines and how to assign events to partitions.	Analytics
Patterns and implementations for a banking cloud transformation	Learn about the patterns and implementations used to transform a banking system for the cloud. A Kafka scaler is used to detect whether the solution needs to activate or deactivate application deployment.	Serverless
Publisher-Subscriber pattern	Learn about the Publisher-Subscriber pattern, which enables an application to announce events to many interested consumers asynchronously. Kafka is recommended for messaging.	Integration
Rate Limiting pattern	Use a rate limiting pattern to avoid or minimize throttling errors. This pattern can implement Kafka for messaging.	Integration
Refactor mainframe applications with Advanced	Learn how to use the automated COBOL refactoring solution from Advanced to modernize your mainframe COBOL applications, run them on Azure, and reduce costs. Kafka can be used as a data source.	Mainframe
Scalable order processing	Learn about a highly scalable, resilient architecture for e-commerce order processing. Event messages enter the system via Kafka and other systems.	Databases
Stream processing with fully managed open-	Stream events by using fully managed Azure data services. Use open-source technologies like Kafka,	Analytics

Architecture	Summary	Technology focus
source data engines	Kubernetes, Cassandra, PostgreSQL, and Redis components.	

Apache MapReduce

Architecture	Summary	Technology focus
Asynchronous messaging options	Learn about asynchronous messaging options in Azure. You can use MapReduce to generate reports on events captured by Event Hubs.	Integration
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. You can use MapReduce for batch processing and to provide functionality for parallel operations in these scenarios.	Databases
Choose a batch processing technology	Learn about technologies for big data batch processing in Azure, including HDInsight with MapReduce.	Analytics
ETL using HDInsight	ETL big data clusters on demand by using HDInsight, Hadoop MapReduce, and Apache Spark.	Analytics
Geode pattern	Deploy back-end services into a set of geographical nodes, each of which can service any client request in any region. This pattern occurs in big data architectures that use MapReduce to consolidate results across machines.	Databases
Minimize coordination	Follow these recommendations to improve scalability by minimizing coordination between application services. Use MapReduce to split work into independent tasks.	Databases

Apache NiFi

Architecture	Summary	Technology focus
Apache NiFi monitoring with MonitoFi	Monitor deployments of Apache NiFi on Azure by using MonitoFi. This tool sends alerts and displays health and performance information in dashboards.	Analytics

Architecture	Summary	Technology focus
Apache NiFi on Azure	Automate data flows with Apache NiFi on Azure. Use a scalable, highly available solution to move data into the cloud or storage and between cloud systems.	Analytics
Helm-based deployments for Apache NiFi	Use Helm charts when you deploy NiFi on AKS. Helm streamlines the process of installing and managing Kubernetes applications.	Analytics
Azure Data Explorer monitoring	Use Azure Data Explorer and NiFi in a hybrid monitoring solution that ingests streamed and batched logs from diverse sources.	Analytics

Apache Oozie

Architecture	Summary	Technology focus
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. You can use Oozie for orchestration in these scenarios.	Databases
Choose a data pipeline orchestration technology	Learn about the key orchestration capabilities of Oozie.	Databases

Apache Solr

Architecture	Summary	Technology focus
Choose a search data store	Learn about the capabilities of search data stores in Azure and the key criteria for choosing one that best matches your needs. Learn about the key capabilities of HDInsight with Solr.	Databases

Apache Spark

Architecture	Summary	Technology focus
Actuarial risk analysis and financial modeling	Learn how an actuarial developer can move an existing solution and its supporting infrastructure to Azure. Use Spark for data analysis or to speed up processing by distributing result aggregation.	Analytics
Advanced analytics	Learn how you can combine any data at any scale with custom machine learning and get near real-time data analytics on streaming services. Use Spark pools to clean and transform structureless datasets and combine them with structured data.	Analytics
AI at the edge with Azure Stack Hub	Bring your trained AI model to the edge with Azure Stack Hub. Integrate it with your applications for low-latency intelligence. Use Spark to store data.	AI
AI at the edge with Azure Stack Hub - disconnected	Take advantage of edge AI when disconnected from the internet and move your AI models to the edge with a solution that includes Azure Stack Hub. Use Spark to store data.	AI
Analytics end-to-end with Azure Synapse	Learn how to use Azure Data Services to build a modern analytics platform capable of handling the most common data challenges. The Spark Pools analytics engine is available from Azure Synapse workspaces.	Analytics
Batch scoring of Spark on Azure Databricks	Build a scalable solution for batch scoring an Apache Spark classification model.	AI
Big data analytics on confidential computing	Use confidential computing on Kubernetes to run big data analytics with Spark inside confidential containers that are protected by Intel Software Guard Extensions.	Databases
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. You can use Spark for batch or stream processing and as an analytical data store.	Databases
Build a content-based recommendation system	Create content-based recommendation systems that can deliver personalized recommendations to your customers by using Spark, Azure Machine Learning, and Azure Databricks.	Analytics
Campaign optimization with HDInsight Spark	Build and deploy a machine learning model to maximize the purchase rate of leads that are targeted by a marketing campaign.	Databases

Architecture	Summary	Technology focus
Choose a batch processing technology	Compare technology choices for big data batch processing in Azure, including options for implementing Spark.	Analytics
Choose a stream processing technology	Compare options for real-time message stream processing in Azure, including options for implementing Spark.	Analytics
Choose an analytical data store	Evaluate analytical data store options for big data in Azure. Learn about the capabilities of Azure Synapse Spark pools.	Analytics
Customer 360 with Azure Synapse and Dynamics 365 Customer Insights	Build an end-to-end Customer 360 solution by using Azure Synapse Analytics and Dynamics 360 Customer Insights. This solution uses Azure Synapse Spark clusters, which can be scaled up and down automatically.	Analytics
Data science and machine learning with Azure Databricks	Improve operations by using Azure Databricks, Delta Lake, and MLflow for data science and machine learning. Develop, train, and deploy machine learning models. Azure Databricks provides managed Spark clusters.	AI
Extract, transform, and load	Learn about extract-transform-load (ETL) and extract-load-transform (ELT) data transformation pipelines and how to use control flows and data flows. In ELT, you can use Spark to query source data. You can also use it together with Hadoop as a data store.	Databases
ETL using HDInsight	ETL big data clusters on demand by using HDInsight, Hadoop MapReduce, and Apache Spark.	Analytics
IoT and data analytics	Build solutions that integrate data from many IoT devices into a comprehensive data analysis architecture to improve and automate decision making. Spark is used to run batch jobs that analyze the data.	Analytics
IoT using Azure Cosmos DB	Learn how to use Azure Cosmos DB to accommodate diverse and unpredictable IoT workloads without sacrificing ingestion or query performance. Azure Databricks, running Spark Streaming, processes event data from devices.	IoT
Loan charge-off predictions with HDInsight Spark	Use HDInsight and machine learning to predict the likelihood of loans getting charged off.	Databases

Architecture	Summary	Technology focus
Many models machine learning with Spark	Learn about many models machine learning in Azure.	AI
Microsoft machine learning products	Compare options for building, deploying, and managing your machine learning models, including the Azure Databricks Spark-based analytics platform and MMLSpark.	AI
Modern data warehouse for small and medium businesses	Use Azure Synapse, Azure SQL Database, and Azure Data Lake Storage to modernize SMB legacy and on-premises data. Tools in the Azure Synapse workspace can use Spark compute capabilities to process data.	Analytics
Natural language processing technology	Choose a natural language processing service for sentiment analysis, topic and language detection, key phrase extraction, and document categorization. Learn about the key capabilities of Azure HDInsight with Spark.	AI
Observability patterns and metrics	Learn how to use observability patterns and metrics to improve the processing performance of a big data system by using Azure Databricks. The Azure Databricks monitoring library streams Spark events and Spark Structured Streaming metrics from jobs.	Databases
Real-time analytics on big data architecture	Get deep-learning analytics and insights from live streaming data. Run advanced analytics on IoT device data and website clickstream logs in near real time. Apache Spark pools clean, transform, and analyze the streaming data and combine it with structured data.	Analytics
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use open-source technologies like Spark, Kafka, Kubernetes, Cassandra, PostgreSQL, and Redis components.	Analytics

Apache Sqoop

Architecture	Summary	Technology focus
Big data architectures	Learn about big data architectures that handle the ingestion, processing, and analysis of data that's too large or complex for traditional database systems. In these scenarios, you can use Sqoop to automate orchestration workflows.	Databases

Architecture	Summary	Technology focus
Choose a data transfer technology	Learn about data transfer options like Azure Import/Export, Data Box, and Sqoop.	Databases

Apache ZooKeeper

Architecture	Summary	Technology focus
Apache NiFi on Azure	Automate data flows with NiFi on Azure. Use a scalable, highly available solution to move data into the cloud or storage and between cloud systems. In this solution, NiFi uses ZooKeeper to coordinate the flow of data.	Analytics
Helm-based deployments for Apache NiFi	Use Helm charts when you deploy NiFi on AKS. Helm streamlines the process of installing and managing Kubernetes applications. In this architecture, ZooKeeper provides cluster coordination.	Analytics
Rate Limiting pattern	Use a rate limiting pattern to avoid or minimize throttling errors. In this scenario, you can use ZooKeeper to create a system that grants temporary leases to capacity.	Integration

Related resources

- Microsoft partner and non-open-source third-party scenarios on Azure
- Scenarios featuring Microsoft on-premises technologies
- Architecture for startups
- Azure and Power Platform scenarios
- Azure and Microsoft 365 scenarios
- Azure and Dynamics 365 scenarios
- Azure for AWS professionals
- Azure for Google Cloud professionals

Open-source scenarios on Azure

Article • 11/01/2023

Microsoft is proud to support open-source projects, initiatives, and foundations and contribute to thousands of open-source communities. By using open-source technologies on Azure, you can run applications your way while optimizing your investments.

This article provides a summary of architectures and solutions that use Azure together with open-source technologies.

For Apache scenarios, see the dedicated article, [Apache scenarios on Azure](#).

BeeGFS

Architecture	Summary	Technology focus
Digital image-based modeling on Azure	Learn how to perform image-based modeling on Azure infrastructure as a service (IaaS) by following the architecture and design guidance in an example scenario. BeeGFS can be used for back-end storage.	Media
Run reservoir simulation software on Azure	Run OPM Flow reservoir simulation and ResInsight visualization software on an Azure HPC compute cluster and visualization VM. BeeGFS is used as an orchestrated parallel file service.	Compute

Chef

Architecture	Summary	Technology focus
Building blocks for autonomous-driving simulation environments	Simulate the behavior of autonomous-driving vehicles. Chef is used to create reusable images that serve as building blocks in the simulation.	Containers
Design a CI/CD pipeline using Azure DevOps	Build a continuous integration and deployment pipeline for a two-tier .NET web application. In this scenario, you can use Chef to implement infrastructure as code or infrastructure as a service.	DevOps

Architecture	Summary	Technology focus
End-to-end governance in Azure	When you use CI/CD pipelines to automate deployments, apply RBAC not just on the Azure Resource Manager side but also earlier in the process when developers check in code. In this scenario, you can use Chef to implement infrastructure as code.	Management

CNCF

Architecture	Summary	Technology focus
Azure Arc hybrid management and deployment for Kubernetes clusters	Learn how Azure Arc extends Kubernetes cluster management and configuration across datacenters, edge locations, and multiple cloud environments. This architecture uses CNCF-certified Kubernetes clusters.	Hybrid
Build CNCF projects by using Azure Kubernetes Service	Learn how to conceptualize, architect, build, and deploy an application that uses projects from the CNCF after deployment of AKS.	Containers
Multicloud blockchain distributed ledger technology (DLT)	See how the open-source Blockchain Automation Framework (BAF) and Azure Arc-enabled Kubernetes work with multiparty DLTs to build a cross-cloud blockchain solution. This architecture uses CNCF-certified Kubernetes clusters.	Blockchain

Elastic

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to Elasticsearch deployments.	Containers
Choose a search data store	Learn about the capabilities of search data stores in Azure, including Elasticsearch.	Databases
Elastic Workplace Search on Azure	Learn how to deploy Elastic Workplace Search to streamline search for your documents and data.	Integration
Microservices architecture on AKS	Learn about the infrastructure and DevOps considerations of deploying and running a	Containers

Architecture	Summary	Technology focus
Monitor a microservices app in AKS	Learn best practices for monitoring a microservices application that runs on AKS, including using Elasticsearch.	Containers
Monitoring and diagnostics guidance	Learn about storing instrumentation data by using technologies like Elasticsearch.	Management

GlusterFS

Architecture	Summary	Technology focus
Digital image-based modeling on Azure	Learn how to perform image-based modeling on Azure IaaS by following the architecture and design guidance in an example scenario. GlusterFS can be used as a storage solution.	Media
SAP S/4HANA in Linux on Azure	Review proven practices for running SAP S/4HANA in a Linux environment on Azure, with high availability. GlusterFS is implemented for a highly available file share.	SAP

Grafana

Architecture	Summary	Technology focus
Apache NiFi monitoring with MonitoFi	Monitor deployments of Apache NiFi on Azure by using MonitoFi. Grafana is used to display data and send alerts.	Analytics
Azure Data Explorer interactive analytics	Use interactive analytics in Azure Data Explorer. Examine structured, semi-structured, and unstructured data with improvised, interactive, fast queries. Use Grafana to build near real-time analytics dashboards.	Analytics
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. Grafana is a core component for providing monitoring and observability in this solution.	Migration

Architecture	Summary	Technology focus
Baseline architecture for an AKS cluster	View a reference architecture for a baseline infrastructure that deploys an AKS cluster. Grafana is recommended as a platform for logging and metrics.	Containers
Build CNCF projects by using Azure Kubernetes Service	Learn how to conceptualize, architect, build, and deploy an application that uses projects from the CNCF. Grafana provides a dashboard for application metrics.	Containers
CI/CD pipeline for container-based workloads	Build a DevOps CI/CD pipeline for a Node.js web app with Jenkins, Azure Container Registry, AKS, Azure Cosmos DB, and Grafana.	Containers
Container CI/CD using Jenkins and Kubernetes on AKS	Get replicable, manageable clusters of containers by orchestrating the deployment of containers with AKS. Grafana displays visualization of infrastructure and application metrics.	DevOps
Content Delivery Network analytics	View an architecture pattern that demonstrates low-latency high-throughput ingestion for large volumes of Azure Content Delivery Network logs for building near real-time analytics dashboards. Grafana can be used to build the dashboards.	Analytics
Enterprise monitoring with Azure Monitor	See an enterprise monitoring solution that uses Azure Monitor to collect and manage data from cloud, on-premises, and hybrid resources. Grafana can be used to build a dashboard for exploring and sharing the data.	DevOps
IoT analytics with Azure Data Explorer	Use Azure Data Explorer for near real-time IoT telemetry analytics on fast-flowing, high-volume streaming data from a wide variety of IoT devices. Use Grafana to build analytics dashboards.	Analytics
JMeter implementation for a load-testing pipeline	Get an overview of an implementation for a scalable cloud load-testing pipeline. The implementation supports use of Grafana for observability on solution components.	Migration
Long-term security log retention with Azure Data Explorer	Store security logs in Azure Data Explorer on a long-term basis. Minimize costs and easily access the data. Use Grafana to build near real-time analytics dashboards.	Analytics
Optimize administration of SQL Server instances in on-	Learn how to use Azure Arc for management, maintenance, and monitoring of SQL Server	Databases

Architecture	Summary	Technology focus
premises and multicloud environments by using Azure Arc	instances in on-premises and multicloud environments. Use Grafana dashboards for monitoring.	
SAP workload automation using SUSE on Azure	Use this solution to bolster productivity and facilitate innovation. Grafana provides monitoring.	SAP
Web application monitoring on Azure	Learn about the monitoring services you can use on Azure by reviewing a reference architecture that uses a dataflow model for use with multiple data sources. Use Azure Monitor Data Source for Grafana to consolidate Azure Monitor and Application Insights metrics.	Web

InfluxDB

Architecture	Summary	Technology focus
Apache NiFi monitoring with MonitoFi	Monitor deployments of Apache NiFi on Azure by using MonitoFi. MonitoFi uses local instances of InfluxDB to provide real-time monitoring and alerts.	Analytics
Monitor a microservices app in AKS	Learn best practices for monitoring a microservices application that runs on AKS. Includes information about using InfluxDB for metrics when data rates trigger throttling.	Microservices

Jenkins

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to CI systems like Jenkins.	Containers
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. A custom Jenkins-based solution is used for CI/CD.	Migration

Architecture	Summary	Technology focus
Building blocks for autonomous-driving simulation environments	Simulate the behavior of autonomous-driving vehicles. Jenkins can be used for CI/CD.	Compute
CI/CD pipeline for container-based workloads	Build a DevOps CI/CD pipeline for a Node.js web app with Jenkins, Azure Container Registry, AKS, Azure Cosmos DB, and Grafana.	Containers
Container CI/CD using Jenkins and Kubernetes on AKS	Get replicable, manageable clusters of containers by orchestrating the deployment of containers with AKS.	DevOps
Design a CI/CD pipeline using Azure DevOps	Build a continuous integration and deployment pipeline for an application. This article focuses on Azure DevOps, but you can use Jenkins as an alternative.	DevOps
End-to-end governance in Azure	When you use CI/CD pipelines to automate deployments, apply RBAC not just on the Azure Resource Manager side but also earlier in the process when developers check in code. This article focuses on Azure DevOps, but you can use Jenkins as an alternative.	Management
Immutable infrastructure CI/CD using Jenkins and Terraform on Azure	When you develop apps, use a continuous integration and continuous deployment (CI/CD) pipeline to automatically push changes to Azure virtual machines.	DevOps
Java CI/CD using Jenkins and Azure Web Apps	Create web apps in Azure App Service. Use the CI/CD pipeline to deliver value to customers faster.	DevOps
MLOps for Python with Azure Machine Learning	Implement a continuous integration (CI), continuous delivery (CD), and retraining pipeline for an AI application by using Azure DevOps and Azure Machine Learning. This solution can be easily adapted for Jenkins.	AI
Run a Jenkins server on Azure	Learn about the architecture and the considerations to take into account when you install and configure Jenkins.	DevOps

Jupyter

Architecture	Summary	Technology focus
Automated Jupyter notebooks for diagnostics	Learn how to automate diagnostic or routine notebooks by using an Azure serverless architecture.	DevOps
Azure Data Explorer interactive analytics	Use interactive analytics in Azure Data Explorer. Examine structured, semi-structured, and unstructured data with improvised, interactive, fast queries. Jupyter Notebook is used to connect to Azure Data Explorer.	Analytics
Choose a data analytics and reporting technology	Evaluate big-data analytics technology options for Azure, including Jupyter Notebook.	Databases
Citizen AI with Power Platform	Learn how to use Azure Machine Learning and Power Platform to quickly create a machine learning proof of concept and production version. Azure Machine Learning provides a hosted Jupyter Notebook environment.	AI
Data analysis in an Azure industrial IoT analytics solution	View an Azure industrial IoT (IIoT) analytics solution. Use visualization, data trends, dashboards, schematic views, and Jupyter Notebook.	IoT
DevOps for quantum computing	Learn about DevOps requirements for quantum-based apps. You can use Jupyter Notebook to develop quantum components.	DevOps
IoT analytics with Azure Data Explorer	Use Azure Data Explorer for near real-time IoT telemetry analytics on fast-flowing, high-volume streaming data from a wide variety of IoT devices. Analyze data that's stored in Azure Data Explorer by using Jupyter Notebook.	Analytics
Machine learning operations (MLOps) framework to upscale machine learning lifecycle with Azure Machine Learning	Learn how to apply the MLOps maturity model to implement a machine learning solution for predicting product shipping levels. Initial experimental models are developed in Jupyter Notebook.	AI
Many models machine learning with Spark	Learn about many models machine learning in Azure. Azure Machine Learning provides a hosted Jupyter Notebook environment.	AI
Precision medicine pipeline with genomics	Build a precision medicine pipeline for genomic analysis and reporting. Use Microsoft Genomics for efficient secondary and tertiary analysis. Jupyter Notebook is used to annotate Microsoft	Analytics

Architecture	Summary	Technology focus
	Genomics output, merge output files with other data, and analyze data.	

KEDA

Architecture	Summary	Technology focus
Azure Functions in a hybrid environment	View an architecture that illustrates how to use Azure Functions from on-premises virtual machines. KEDA provides event-driven scale in Kubernetes clusters.	Serverless
AKS in event stream processing	View a serverless event-driven architecture that runs on AKS with a KEDA scaler. The solution ingests and processes a stream of data and writes the results to a database.	Containers
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. KEDA is used to scale a service that processes funds transfers.	Containers
Baseline architecture for an AKS cluster	View a reference architecture for a baseline infrastructure that deploys an AKS cluster. In this scenario, you can use KEDA to scale event-driven workloads.	Containers
Integrate Event Hubs with Azure Functions	Learn how to architect, develop, and deploy efficient and scalable code that runs on Azure Functions and responds to Azure Event Hubs events. KEDA scaler for Event Hubs can be used for Kubernetes hosted apps.	Serverless
Patterns and implementations for a banking cloud transformation	Learn about the patterns and implementations used to transform a banking system for the cloud. Includes an architecture for KEDA scaling.	Serverless

Kubernetes

Architecture	Summary	Technology focus
Advanced AKS microservices architecture	Learn about a scalable, highly secure Azure Kubernetes Service (AKS) microservices architecture that builds on recommended AKS microservices baseline architectures and implementations.	Containers
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications.	Containers
Azure Arc hybrid management and deployment for Kubernetes clusters	Learn how Azure Arc extends Kubernetes cluster management and configuration across datacenters, edge locations, and multiple cloud environments.	Hybrid
Azure Kubernetes in event stream processing	View a serverless event-driven architecture that runs on AKS with a KEDA scaler. The solution ingests and processes a stream of data and writes the results to a database.	Containers
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. The solution uses Kubernetes clusters.	Containers
Baseline architecture for an AKS cluster	View a reference architecture for a baseline infrastructure that deploys an AKS cluster.	Containers
Big data analytics on confidential computing	Use confidential computing on Kubernetes to run big data analytics with Spark inside confidential containers that are protected by Intel Software Guard Extensions.	Analytics
Build a CI/CD pipeline for microservices on Kubernetes	Learn about building a CI/CD pipeline for deploying microservices to AKS.	Microservices
Build CNCF projects by using Azure Kubernetes Service	Learn how to conceptualize, architect, build, and deploy an application that uses projects from the CNCF after deployment of AKS.	Containers
Choose a bare-metal Kubernetes-at-the-edge platform option	Find the best option for configuring Kubernetes clusters at the edge.	Containers
Choose a Kubernetes-at-the-edge compute option	Learn about trade-offs for various options that are available for extending compute on the edge.	Containers

Architecture	Summary	Technology focus
Choose an Azure multiparty computing service	Decide which multiparty computing services to use for your solution. Includes information about using Kubernetes to manage containers.	Blockchain
Container CI/CD using Jenkins and Kubernetes on AKS	Get replicable, manageable clusters of containers by orchestrating the deployment of containers with AKS.	DevOps
Container orchestration for microservices	Learn how container orchestration makes it easy to manage complex multi-container microservice deployments, scaling, and cluster health. Review options for microservices container orchestration, including AKS.	Microservices
Create a CI/CD pipeline for AI apps using Azure Pipelines, Docker, and Kubernetes	Create a continuous integration and continuous delivery pipeline for AI applications by using Docker and Kubernetes.	AI
Employee retention with Databricks and Kubernetes	Learn how to use Kubernetes to build, deploy, and monitor a machine learning model for employee attrition that can be integrated with external applications.	Analytics
GitOps for Azure Kubernetes Service	See a GitOps solution for an AKS cluster. This solution provides full audit capabilities, policy enforcement, and early feedback.	Containers
Helm-based deployments for Apache NiFi	Use Helm charts when you deploy NiFi on AKS. Helm streamlines the process of installing and managing Kubernetes applications.	Analytics
Microservices architecture on AKS	Learn about the infrastructure and DevOps considerations of deploying and running a microservices architecture on AKS.	Containers
CI/CD for AKS apps with Azure Pipelines	Learn how AKS simplifies the deployment and management of microservices-based architecture.	Containers
Patterns and implementations for a banking cloud transformation	Learn about the patterns and implementations used to transform a banking system for the cloud. Includes an architecture for Kubernetes Event-driven Autoscaler (KEDA) scaling.	Serverless
Use Application Gateway Ingress Controller with a multitenant AKS cluster	Learn how to use the Application Gateway Ingress Controller with your AKS cluster to expose microservice-based applications to the internet.	Containers

Architecture	Summary	Technology focus
Use Azure Firewall to help protect an AKS cluster	Deploy an AKS cluster in a hub-and-spoke network topology by using Terraform and Azure DevOps. Help protect the inbound and outbound traffic by using Azure Firewall.	Containers

Lustre

Architecture	Summary	Technology focus
Digital image-based modeling on Azure	Learn how to perform image-based modeling on Azure IaaS by following the architecture and design guidance in an example scenario. Lustre can be used as a storage solution.	Media
Run reservoir simulation software on Azure	Run OPM Flow reservoir simulation and ResInsight visualization software on an Azure HPC compute cluster and visualization VM. Lustre is used as an orchestrated parallel file service.	Compute
SAS on Azure architecture	Learn how to run SAS analytics products on Azure. In this scenario, Lustre is a recommended option for permanent storage.	Compute

MariaDB

Architecture	Summary	Technology focus
Core startup stack architecture	Review the components of a simple core startup stack architecture. Azure Database for MariaDB is one recommended relational database.	Startup
Mainframe and midrange data replication to Azure using Qlik	Use Qlik Replicate to migrate mainframe and midrange systems to the cloud, or to extend such systems with cloud applications. Azure Database for MariaDB is one recommended relational database.	Mainframe
Mainframe file replication and sync on Azure	Learn about several options for moving, converting, transforming, and storing mainframe and midrange file system data on-premises and in Azure. Store data in Azure Database for MariaDB.	Mainframe

Architecture	Summary	Technology focus
Modernize mainframe and midrange data	Learn how to modernize IBM mainframe and midrange data and see how to use a data-first approach to migrate this data to Azure. Azure Database for MariaDB is one recommended relational database.	Mainframe
Replicate and sync mainframe data in Azure	Replicate data while modernizing mainframe and midrange systems. Sync on-premises data with Azure data during modernization. Azure Database for MariaDB is one recommended relational database.	Mainframe
Scalable and secure WordPress on Azure	Learn how to use Content Delivery Network and other Azure services to deploy a highly scalable and highly secure installation of WordPress. In this scenario, MariaDB is used as a data store.	Web
Understand data store models	Learn about the high-level differences between the various data storage models found in Azure data services. Azure Database for MariaDB is one example of a relational database.	Databases

MLflow

Architecture	Summary	Technology focus
Data science and machine learning with Azure Databricks	Improve operations by using Azure Databricks, Delta Lake, and MLflow for data science and machine learning. Develop, train, and deploy machine learning models.	AI
Determine customer lifetime value and churn with Azure AI services	Learn how to create a solution for predicting customer lifetime value and churn by using Azure Machine Learning. The solution demonstrates how to use MLflow to track machine learning experiments.	AI
Employee retention with Databricks and Kubernetes	Learn how to use Kubernetes to build, deploy, and monitor a machine learning model for employee attrition that can be integrated with external applications. Includes a proof-of-concept that illustrates how to train an MLflow model for employee attrition on Azure Databricks.	Analytics
Modern analytics architecture with Azure Databricks	Create a modern analytics architecture with Azure Databricks, Data Lake Storage, and other Azure services. Unify data, analytics, and AI workloads at any scale.	Analytics

Architecture	Summary	Technology focus
	MLflow manages parameter, metric, and machine learning model tracking.	
Orchestrate MLOps on Azure Databricks using Databricks notebooks	Learn about an approach to MLOps that involves running model training and batch scoring on Azure Databricks by using Azure Databricks notebooks for orchestration. MLflow manages the machine learning lifecycle.	AI
Population health management for healthcare	Use population health management to improve clinical and health outcomes and reduce costs. The Azure Machine Learning native support for MLflow is used to log experiments, store models, and deploy models.	AI

Moodle

Architecture	Summary	Technology focus
Moodle deployment with Azure NetApp Files	Deploy Moodle with Azure NetApp Files for a resilient solution that offers high-throughput, low-latency access to scalable shared storage.	Storage

MySQL

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to systems that run MySQL database workloads.	Containers
Build CNCF projects by using Azure Kubernetes Service	Learn how to conceptualize, architect, build, and deploy an application that uses projects from the CNCF after deployment of AKS. MySQL is used to store expense reports.	Containers
Finance management apps with Azure Database for MySQL	Use Azure Database for MySQL to store critical data with high security and provide users with high-value analytics and insights on aggregated data.	Databases
IBM z/OS online transaction processing on Azure	Migrate a z/OS online transaction processing (OLTP) workload to an Azure application that's cost-effective,	Mainframe

Architecture	Summary	Technology focus
	responsive, scalable, and adaptable. The data layer can include Azure implementations of MySQL databases.	
Intelligent apps using Azure Database for MySQL	Use Azure Database for MySQL to develop sophisticated machine learning and visualization apps that provide analytics and information that you can act on.	Databases
Java CI/CD using Jenkins and Azure Web Apps	Use App Service to create web apps backed by Azure Database for MySQL. Use the CI/CD pipeline to deliver value to customers faster.	DevOps
Lift and shift to containers with AKS	Migrate existing applications to containers in AKS. Use Open Service Broker for Azure to access databases like Azure Database for MySQL.	Containers
Mainframe file replication and sync on Azure	Learn about several options for moving, converting, transforming, and storing mainframe and midrange file system data on-premises and in Azure. Store data in Azure Database for MySQL.	Mainframe
CI/CD for AKS apps with Azure Pipelines	Learn how AKS simplifies the deployment and management of microservices-based architecture. Use Azure Database for MySQL to store and retrieve information used by the microservices.	Containers
Online transaction processing (OLTP)	Learn about atomicity, consistency, and other features of OLTP, which manages transactional data and supports querying. Azure Database for MySQL is one Azure data store that meets the requirements for OLTP.	Databases
Scalable apps using Azure Database for MySQL	Use Azure Database for MySQL to rapidly build engaging, performant, and scalable cross-platform and native apps for iOS, Android, Windows, or Mac.	Mobile
Security considerations for highly sensitive IaaS apps in Azure	Learn about VM security, encryption, NSGs, perimeter networks (also known as DMZs), access control, and other security considerations for highly sensitive IaaS and hybrid apps. A common replication scenario for IaaS architectures uses MySQL Replication.	Security
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use open-source technologies like Kafka, Kubernetes, Cassandra, MySQL, and Redis components.	Analytics
Understand data store models	Learn about the high-level differences between the various data storage models found in Azure data	Databases

Architecture	Summary	Technology focus
	services. Azure Database for MySQL is one example of a relational database.	

PostgreSQL

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to systems that run PostgreSQL database workloads.	Containers
Azure Database for PostgreSQL intelligent apps	Use Azure Database for PostgreSQL to develop sophisticated machine learning and visualization apps that provide analytics and information that you can act on.	Databases
Build a telehealth system on Azure	Learn how to build a telehealth system that connects a professional healthcare organization to its remote patients. Azure Database for PostgreSQL stores user and device-related data.	Databases
Data cache	Store and share database query results, session states, static contents, and more by using a common cache-aside pattern. This solution works with data stored in Azure Database for PostgreSQL and other databases.	Databases
Data streaming with AKS	Use AKS to easily ingest and process a real-time data stream with millions of data points collected via sensors. Processed data is stored in Azure Database for PostgreSQL.	Containers
Finance management apps using Azure Database for PostgreSQL	Use Azure Database for PostgreSQL to store critical data with high security and provide users with high-value analytics and insights on aggregated data.	Databases
Geospatial data processing and analytics	Collect, process, and store geospatial data by using managed Azure services. Make the data available through web apps. Visualize, explore, and analyze the data. In this solution, Azure Database for PostgreSQL stores GIS data.	Analytics
IBM z/OS online transaction processing on Azure	Migrate a z/OS OLTP workload to an Azure application that's cost-effective, responsive, scalable,	Mainframe

Architecture	Summary	Technology focus
	and adaptable. The data layer can include Azure Database for PostgreSQL.	
Integrate IBM mainframe and midrange message queues with Azure	Learn about a data-first approach to middleware integration that enables IBM message queues. This approach supports Azure Database for PostgreSQL.	Mainframe
Mainframe file replication and sync on Azure	Learn about several options for moving, converting, transforming, and storing mainframe and midrange file system data on-premises and in Azure. PostgreSQL is the main database for a common AIML use case.	Mainframe
Online transaction processing (OLTP)	Learn about atomicity, consistency, and other features of OLTP, which manages transactional data and supports querying. Azure Database for PostgreSQL is one Azure data store that meets the requirements for OLTP.	Databases
Oracle database migration: Refactor	Refactor your Oracle database by using Azure Database Migration Service, and move it to PostgreSQL.	Migration
Overview of Oracle database migration	Learn about Oracle database migration paths and the methods you can use to migrate your schema to SQL or PostgreSQL.	Migration
Scalable apps using Azure Database for PostgreSQL	Use Azure Database for PostgreSQL to rapidly build engaging, performant, and scalable cross-platform and native apps for iOS, Android, Windows, or Mac.	Mobile
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use open-source technologies like Kafka, Kubernetes, Cassandra, PostgreSQL, and Redis components.	Analytics
Understand data store models	Learn about the high-level differences between the various data storage models found in Azure data services. Azure Database for PostgreSQL is one example of a relational database.	Databases
Use LzLabs Software Defined Mainframe (SDM) in an Azure VM deployment	Learn an approach for rehosting mainframe legacy applications in Azure by using the LzLabs SDM platform. This architecture uses PostgreSQL IaaS.	Mainframe

Prometheus

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. Astra Trident provides a rich set of Prometheus metrics that you can use to monitor provisioned storage.	Containers
Architecture of an AKS regulated cluster for PCI-DSS 3.2.1	Learn about an architecture for an AKS cluster that runs a workload in compliance with the Payment Card Industry Data Security Standard. Prometheus metrics are used in monitoring.	Containers
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. Prometheus is a core component for monitoring test results.	Migration
Baseline architecture for an AKS cluster	View a reference architecture for a baseline infrastructure that deploys an AKS cluster. Azure Monitor, the recommended monitoring tool, can be used to visualize Prometheus metrics.	Containers
Build CNCF projects by using Azure Kubernetes Service	Learn how to conceptualize, architect, build, and deploy an application that uses projects from the CNCF. Prometheus captures application metrics.	Containers
JMeter implementation for a load-testing pipeline	Get an overview of an implementation for a scalable cloud load-testing pipeline. The implementation supports use of Prometheus for observability on solution components.	Migration
Microservices architecture on AKS	Learn about the infrastructure and DevOps considerations of deploying and running a microservices architecture on AKS. Prometheus can be used for cluster monitoring.	Containers
Monitor a microservices app in AKS	Learn best practices for monitoring a microservices application that runs on AKS. Includes information about using Prometheus for metrics when data rates trigger throttling.	Containers
SAP workload automation using SUSE on Azure	Use this solution to bolster productivity and facilitate innovation. Prometheus provides monitoring.	SAP

PyTorch

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to systems that run AI and machine learning components like PyTorch.	Containers
Data science and machine learning with Azure Databricks	Improve operations by using Azure Databricks, Delta Lake, and MLflow for data science and machine learning. Azure Databricks uses pre-installed, optimized machine learning frameworks, including PyTorch.	AI
Machine learning in IoT Edge vision	Learn about machine learning data and models in Azure IoT Edge vision AI solutions. Includes information about PyTorch.	IoT

RabbitMQ

Architecture	Summary	Technology focus
Automated guided vehicles fleet control	Learn about an end-to-end approach for an automotive original equipment manufacturer (OEM). Includes a reference architecture and several published open-source libraries that you can reuse. RabbitMQ is used as a message broker.	Web
Publisher-Subscriber pattern	Learn about the Publisher-Subscriber pattern, which enables an application to announce events to many interested consumers asynchronously. RabbitMQ is recommended for messaging.	Integration

Red Hat

Architecture	Summary	Technology focus
AIX UNIX on-premises to Azure Linux migration	Migrate an on-premises IBM AIX system and web application to a highly available, highly secure Red Hat Enterprise Linux solution in Azure.	Mainframe
Banking system cloud	Use simulated and actual applications and existing workloads to monitor the reaction of a solution	Containers

Architecture	Summary	Technology focus
transformation on Azure	infrastructure for scalability and performance. Azure Red Hat OpenShift is used for autoscaling tests.	
Container orchestration for microservices	Learn how container orchestration makes it easy to manage complex multi-container microservice deployments, scaling, and cluster health. Review options for microservices container orchestration, including Azure Red Hat OpenShift.	Microservices
JBoss deployment with Red Hat on Azure	Learn how the Red Hat JBoss Enterprise Application Platform (JBoss EAP) streamlines and simplifies the development and deployment of a range of applications.	Containers
Run a Linux VM on Azure	Learn best practices for running a Linux virtual machine on Azure. Azure supports many popular Linux distributions, including Red Hat Enterprise.	Compute
SAP HANA for Linux VMs in scale-up systems	Learn proven practices for running SAP HANA in a high-availability, scale-up environment that supports disaster recovery. Use Red Hat Enterprise Linux in multi-node configurations. For high availability, use a Pacemaker cluster on Red Hat Enterprise Linux.	SAP
SAP S/4HANA in Linux on Azure	Learn proven practices for running SAP S/4HANA in a Linux environment on Azure, with high availability. Red Hat Enterprise Linux is used for a high availability SAP Central Services cluster.	SAP
SAS on Azure	Learn how to run SAS analytics products on Azure. SAS supports Red Hat 7 and later.	Compute
SWIFT's AMH with Alliance Connect Virtual	Run SWIFT AMH on Azure. Use this messaging solution with the Alliance Connect Virtual networking solution, which also runs on Azure. A key component, the AMH node, runs on JBoss EAP on Red Hat Enterprise Linux.	Networking

Redis

Architecture	Summary	Technology focus
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. Azure Cache for Redis is used in a Publish-Subscribe messaging pattern for bank transactions.	Containers

Architecture	Summary	Technology focus
COVID-19 safe solutions with IoT Edge	Create a COVID-19 safe environment that monitors social distance, mask/PPE use, and occupancy requirements with CCTVs and Azure IoT Edge, Azure Stream Analytics, and Azure Machine Learning. Redis is used to store cloud data for analytics and visualization.	IoT
Data cache	Azure Cache for Redis provides a cost-effective solution to scale read and write throughput of your data tier. Store and share database query results, session states, static contents, and more by using a common cache-aside pattern.	Databases
Data streaming with AKS	Use AKS to easily ingest and process a real-time data stream with millions of data points collected via sensors. Azure Cache for Redis is used to cache processed data.	Containers
DevTest and DevOps for PaaS solutions	Combine Azure platform as a service (PaaS) resources with DevTest and DevOps practices to support rapid iteration cycles and reduced overhead. Azure Cache for Redis provides an in-memory data store that's provisioned by Terraform.	DevOps
Messaging	Learn how Azure Cache for Redis routes real-time messages in publish and subscribe systems.	Databases
Non-relational data and NoSQL	Learn about non-relational databases that store data as key-value pairs, graphs, time series, objects, and other storage models, based on data requirements. Azure Cache for Redis can be used to store key-value pairs.	Databases
Personalized offers	Build intelligent marketing systems that provide customer-tailored content by using machine learning models that analyze data from multiple sources. Azure Cache for Redis is used to provide pre-computed product affinities for customers.	AI
Publisher-Subscriber pattern	Learn about the Publisher-Subscriber pattern, which enables an application to announce events to many interested consumers asynchronously. In this pattern, Redis can be used for messaging.	Integration
Rate Limiting pattern	Use a rate limiting pattern to avoid or minimize throttling errors. In this scenario, you can use Redis/Redsync to create a system that grants temporary leases to capacity.	Integration
Re-engineer mainframe batch	Use Azure services to re-engineer mainframe batch applications. This architecture change can reduce costs	Mainframe

Architecture	Summary	Technology focus
applications on Azure	and improve scalability. You can use Azure Cache for Redis to speed up a re-engineered application.	
Scalable Sitecore marketing website	Learn how the Sitecore Experience Platform (XP) provides the data, integrated tools, and automation you need to engage customers throughout an iterative lifecycle. In this solution, Sitecore's session state is managed by Azure Cache for Redis.	Web
Scalable web apps with Azure Redis Cache	Improve app performance by using Azure Cache for Redis to improve responsiveness and handle increasing loads with fewer web-compute resources.	Web
Baseline web application with zone redundancy	Use the proven practices in this reference architecture to improve redundancy, scalability and performance in an Azure App Service web application.	Web
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use open-source technologies like Kafka, Kubernetes, Cassandra, PostgreSQL, and Redis components.	Analytics

SUSE

Architecture	Summary	Technology focus
Run SAP BW/4HANA with Linux VMs	Learn about the SAP BW/4HANA application tier and how it's suitable for a high availability, small-scale production environment of SAP BW/4HANA on Azure. Azure is certified to run SAP BW/4HANA on SUSE Linux Enterprise.	SAP
SAP deployment in Azure using an Oracle database	Learn proven practices for running SAP on Oracle in Azure, with high availability. In this architecture, SUSE SBD can be used as part of a mechanism to automate failovers.	SAP
SAP HANA for Linux VMs in scale-up systems	Learn proven practices for running SAP HANA in a high availability, scale-up environment that supports disaster recovery on Azure. For high availability, use a Pacemaker cluster on SUSE Linux Enterprise Server.	SAP
SAP S/4HANA in Linux on Azure	Review proven practices for running SAP S/4HANA in a Linux environment on Azure, with high availability. SUSE Linux Enterprise Server is used for a high availability SAP Central Services cluster.	SAP

Architecture	Summary	Technology focus
SAP workload automation using SUSE on Azure	Use this solution to bolster productivity and facilitate innovation.	SAP
SAS on Azure architecture	Learn how to run SAS analytics products on Azure. SAS supports SUSE Linux Enterprise Server (SLES) 12.2.	Compute

TensorFlow

Architecture	Summary	Technology focus
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to systems that run AI and machine learning components like TensorFlow.	Containers
Data science and machine learning with Azure Databricks	Improve operations by using Databricks, Delta Lake, and MLflow for data science and machine learning. Develop, train, and deploy machine learning models. Azure Databricks uses pre-installed, optimized machine learning frameworks, including TensorFlow.	AI
Machine learning in IoT Edge vision	Learn about machine learning data and models in Azure IoT Edge vision AI solutions. Includes information about TensorFlow.	IoT
Vision classifier model with Azure Cognitive Services Custom Vision	Create an image classifier with a solution architecture that includes Microsoft AirSim Drones simulator, Azure Cognitive Services Custom Vision, and TensorFlow.	AI

Terraform

Architecture	Summary	Technology focus
Architectural approaches for the deployment and configuration of multitenant solutions	Learn about approaches to consider when you deploy and configure a multitenant solution. Terraform is recommended for automation.	Multitenancy
Automated guided vehicles fleet control	Learn about an end-to-end approach for an automotive original equipment manufacturer	Web

Architecture	Summary Includes a reference architecture and several published open-source libraries that you can reuse. Terraform is used to deploy Azure instances.	Technology focus
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. Terraform is used for load testing.	Migration
Deployment Stamps pattern	Learn about the Deployment Stamps pattern, which deploys many independent copies of application components. Terraform is recommended for deployment.	Networking
Design a CI/CD pipeline using Azure DevOps	Build a continuous integration and deployment pipeline for an application. In this architecture, Terraform can be used for deployment.	DevOps
DevSecOps on AKS	Learn about DevSecOps, a solution that incorporates security best practices from the beginning of development. Terraform is used to manage infrastructure as code.	DevOps
DevTest and DevOps for PaaS solutions	Combine Azure PaaS resources with DevTest and DevOps practices to support rapid iteration cycles and reduced overhead. Terraform provisions and modifies resources for the environments.	DevOps
End-to-end governance in Azure	When you use CI/CD pipelines to automate deployments, apply RBAC not just on the Azure Resource Manager side but also earlier in the process when developers check in code. The scenario described uses Terraform for infrastructure as code.	Management
Gridwich cloud media system	Learn about a stateless action execution workflow that ingests, processes, and delivers media assets using Terraform Sandwiches and Event Grid Sandwiches.	Media
Gridwich CI/CD pipeline	Learn about the guiding principles and considerations for the Gridwich continuous CD/CD pipeline, including information about Terraform.	Media
Gridwich keys and secrets management	Learn about the two types of keys Gridwich uses, and logic apps that add, change, or rotate the keys. Terraform is used in an app that rotates or adds third-party keys.	Media

Architecture	Summary	Technology focus
Gridwich Media Services setup and scaling	Learn how Gridwich uses Azure Media Services V2 and V3 APIs to set up authentication and authorization, and how to scale Media Services resources for media processing. A Terraform file is used in the authentication and authorization process.	Media
Gridwich pipeline-generated admin scripts	Learn about Gridwich pipeline-generated admin scripts and how to run them. The pipelines use Terraform to generate and publish the scripts.	Media
Gridwich variable flow	Learn how Gridwich converts Azure Pipelines pipeline variable group variables to Terraform variables.	Media
Immutable infrastructure CI/CD using Jenkins and Terraform on Azure	When you develop apps, use a continuous integration and continuous deployment (CI/CD) pipeline to automatically push changes to Azure virtual machines.	DevOps
JMeter implementation for a load-testing pipeline	Get an overview of an implementation for a scalable cloud load-testing pipeline. The implementation uses JMeter and Terraform to provision and remove the required infrastructure.	Migration
SAP workload automation using SUSE on Azure	Use this solution to bolster productivity and facilitate innovation. Terraform is used to deploy the SAP infrastructure into Azure.	SAP
Use Azure Firewall to help protect an AKS cluster	Deploy an AKS cluster in a hub-and-spoke network topology by using Terraform and Azure DevOps. Help protect inbound and outbound traffic by using Azure Firewall.	Containers
Virtual network integrated serverless microservices	Learn about an end-to-end solution for health records management that uses Azure Functions microservices integrated with other services via a virtual network. Terraform automates all code and infrastructure deployments.	Security

Umbraco

Architecture	Summary	Technology focus
Scalable Umbraco CMS web app	Run an Umbraco content management system on the Web Apps feature of App Service. Use Azure managed services for a high availability environment.	Web

WordPress

Architecture	Summary	Technology focus
Scalable and secure WordPress on Azure	Learn how to use Azure Content Delivery Network and other Azure services to deploy a highly scalable and highly secure installation of WordPress.	Web

Related resources

- Microsoft partner and non-open-source third-party scenarios on Azure
- Scenarios featuring Microsoft on-premises technologies
- Architecture for startups
- Azure and Power Platform scenarios
- Azure and Microsoft 365 scenarios
- Azure and Dynamics 365 scenarios
- Azure for AWS professionals
- Azure for Google Cloud professionals

Microsoft partner and third-party scenarios on Azure

Article • 06/14/2023

This article explores Microsoft partner and third-party, non-open source scenarios on Microsoft Azure.

Microsoft partners make up a community of organizations that work with Microsoft to create innovative solutions for you. Driven by the opportunities of the intelligent cloud, Microsoft is prioritizing investments that support these opportunities.

The [Azure Sponsorship for ISVs program](#) helps independent software vendors (ISVs) use Azure services to drive platform innovation and develop new solutions that can accelerate your digital transformation.

Visit [Azure Marketplace](#) to discover, try, and deploy cloud software from Microsoft and Microsoft partners.

This article provides a summary of architectures and solutions that use Azure together with partner and third-party solutions.

We also recommend you browse our open-source solutions for Microsoft Azure:

- [Apache open-source scenarios on Azure](#)
- [Open-source scenarios on Azure](#)

Advanced

Architecture	Summary	Technology focus
Refactor mainframe applications with Advanced	Learn how to use the automated COBOL refactoring solution from Advanced to modernize your mainframe COBOL applications, run them on Azure, and reduce costs.	Mainframe

Astadia

Architecture	Summary	Technology focus
--------------	---------	------------------

Architecture	Summary	Technology focus
Unisys Dorado mainframe migration to Azure with Astadia & Micro Focus	Migrate Unisys Dorado mainframe systems with Astadia and Micro Focus products. Move to Azure without rewriting code, switching data models, or updating screens.	Mainframe

Avanade

Architecture	Summary	Technology focus
IBM z/OS mainframe migration with Avanade AMT	Learn how to use the Avanade Automated Migration Technology (AMT) framework to migrate IBM z/OS mainframe workloads to Azure.	Mainframe
Unisys mainframe migration with Avanade AMT	Learn options for using the AMT framework to migrate Unisys mainframe workloads to Azure.	Mainframe

CluedIn

Architecture	Summary	Technology focus
Master Data Management with Azure and CluedIn	Use CluedIn eventual connectivity data integration to blend data from many siloed data sources and prepare it for analytics and business operations.	Databases
Migrate master data services to Azure with CluedIn and Azure Purview	Use CluedIn to migrate your master data services solution to Azure by using CluedIn and Azure Purview.	Databases

Confluent

Architecture	Summary	Technology focus
Banking system cloud transformation on Azure	Use simulated and actual applications and existing workloads to monitor the reaction of a solution infrastructure for scalability and performance. Kafka is used with Confluent Schema Registry for streaming.	Migration

Architecture	Summary	Technology focus
Real-time processing	Use real-time processing solutions to capture data streams and generate reports or automated responses with minimal latency. Kafka, which is available via ConfluentCloud, is recommended for real-time message ingestion.	Databases

Couchbase

Architecture	Summary	Technology focus
High availability in Azure public MEC	Learn how to deploy workloads in active-standby mode to achieve high availability and disaster recovery in Azure public multi-access edge compute. Couchbase can provide IaaS services that support geo-replication.	Hybrid

Double-Take

Architecture	Summary	Technology focus
SMB disaster recovery with Azure Site Recovery	Learn how small and medium-sized businesses can inexpensively implement cloud-based disaster recovery solutions by using Azure Site Recovery or Double-Take DR.	Management
SMB disaster recovery with Double-Take DR	Learn how small and medium-sized businesses can inexpensively implement cloud-based disaster recovery solutions by using a partner solution like Double-Take DR.	Management

Episerver

Architecture	Summary	Technology focus
Scalable Episerver marketing website	Run multi-channel digital marketing websites on one platform. Start and stop campaigns on demand. Manage site and campaign performance by using Episerver.	Web

Gremlin

Architecture	Summary	Technology focus
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use technologies like Kafka, Kubernetes, Gremlin, PostgreSQL, and Redis components.	Analytics

Infinite i

Architecture	Summary	Technology focus
IBM System i (AS/400) to Azure using Infinite i	Use Infinite i to easily migrate your IBM System i (AS/400) workloads to Azure. You can lower costs, improve performance, improve availability, and modernize.	Mainframe

LzLabs

Architecture	Summary	Technology focus
Use LzLabs Software Defined Mainframe (SDM) in an Azure VM deployment	Learn an approach for rehosting mainframe legacy applications in Azure by using the LzLabs SDM platform.	Mainframe

Micro Focus

Architecture	Summary	Technology focus
Micro Focus Enterprise Server on Azure VMs	Optimize, modernize, and streamline IBM z/OS mainframe applications by using Micro Focus Enterprise Server 6.0 on Azure VMs.	Mainframe
Unisys Dorado mainframe migration to Azure with Astadia & Micro Focus	Migrate Unisys Dorado mainframe systems with Astadia and Micro Focus products. Move to Azure without rewriting code, switching data models, or updating screens.	Mainframe

MongoDB

Architecture	Summary	Technology focus
Advanced AKS microservices architecture	Learn about a scalable, highly secure AKS microservices architecture that builds on recommended AKS microservices baseline architectures and implementations. In this architecture, Azure Cosmos DB stores data by using the open-source Azure Cosmos DB for MongoDB.	Containers
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications. This solution applies to systems that run MongoDB database workloads.	Containers
Core startup stack architecture	Review the components of a simple core startup stack architecture. MongoDB is recommended for uses cases that require a NoSQL database.	Startup
COVID-19 safe solutions with IoT Edge	Create a COVID-19 safe environment that monitors social distance, mask/PPE use, and occupancy requirements with CCTVs and IoT Edge, Stream Analytics, and Azure Machine Learning. MongoDB is used to store cloud data for Power BI analytics and visualizations.	IoT
Data considerations for microservices	Learn about managing data in a microservices architecture. The MongoDB API is used with Azure Cosmos DB in an example scenario.	Microservices
High availability in Azure public MEC	Learn how to deploy workloads in active-standby mode to achieve high availability and disaster recovery in Azure public multiaccess edge compute. MongoDB can provide IaaS services that support geo-replication.	Hybrid
Baseline web application with zone redundancy	Use the proven practices in this reference architecture to improve redundancy, scalability and performance in an App Service web application. MongoDB is recommended for non-relational data.	Web
Stream processing with fully managed open-source data engines	Stream events by using fully managed Azure data services. Use open-source technologies like Kafka, Kubernetes, MongoDB, PostgreSQL, and Redis components.	Analytics

Architecture	Summary	Technology focus
Virtual network integrated serverless microservices	Learn about an end-to-end solution for health records management that uses Azure Functions microservices integrated with other services via a virtual network. In this solution, microservices store data in Azure Cosmos DB, using the MongoDB Node.js driver.	Security

NetApp

Architecture	Summary	Technology focus
AIX UNIX on-premises to Azure Linux migration	Migrate an on-premises IBM AIX system and web application to a highly available, highly secure Red Hat Enterprise Linux solution in Azure. Azure NetApp Files provides shared NAS.	Mainframe
Application data protection for AKS workloads on Azure NetApp Files	Deploy Astra Control Service with Azure NetApp Files for data protection, disaster recovery, and mobility for AKS applications.	Storage
SAP workload development and test settings	Learn how to establish non-production development and test environments for SAP NetWeaver in a Windows or Linux environment on Azure. Azure NetApp Files is recommended for storage of SAP executables and HANA data and logs.	SAP
Enterprise file shares with disaster recovery	Learn how to implement resilient NetApp file shares. Failure of the primary Azure region causes automatic failover to the secondary Azure region.	Storage
FSLogix configuration examples	Learn how to build virtual desktop infrastructure solutions at enterprise scale by using FSLogix. Azure NetApp Files is recommended for storing profiles.	Hybrid
General mainframe refactor to Azure	Learn how to refactor mainframe applications to run more cost-effectively and efficiently on Azure. Azure NetApp Files is recommended for file storage.	Mainframe
Moodle deployment with Azure NetApp Files	Deploy Moodle with Azure NetApp Files for a resilient solution that offers high-throughput, low-latency access to scalable shared storage.	Storage

Architecture	Summary	Technology focus
Multiple forests with AD DS and Azure AD	Learn how to create multiple Active Directory forests with Azure Virtual Desktop. Azure NetApp Files is one recommended storage solution for the scenario.	Virtual Desktop
Oracle Database with Azure NetApp Files	Implement a high-bandwidth, low-latency solution for Oracle Database workloads. Use Azure NetApp Files to get enterprise-scale performance and to reduce costs.	Storage
Refactor mainframe computer systems that run Adabas & Natural	Learn how to modernize mainframe computer systems that run Adabas & Natural and move them to the cloud. Azure NetApp Files is used to store persistent data.	Mainframe
Run SAP BW/4HANA with Linux VMs	Learn about the SAP BW/4HANA application tier and how it's suitable for a high-availability, small-scale production environment of SAP BW/4HANA on Azure. Azure NetApp Files is used by a high-availability cluster for shared file storage.	SAP
SAP deployment in Azure using an Oracle database	Learn proven practices for running SAP on Oracle in Azure, with high availability.	SAP
SAP HANA for Linux VMs in scale-up systems	Learn proven practices for running SAP HANA in a high availability scale-up environment that supports disaster recovery.	SAP
SAP S/4HANA in Linux on Azure	Learn proven practices for running SAP S/4HANA in a Linux environment on Azure, with high availability.	SAP
SAS on Azure architecture	Learn how to run SAS analytics products on Azure. Includes recommendations for using Azure NetApp Files.	Compute
SQL Server on Azure Virtual Machines with Azure NetApp Files	Implement a high-bandwidth, low-latency solution for SQL Server workloads. Use Azure NetApp Files to get enterprise-scale performance and to reduce costs.	Storage

Oracle

Architecture	Summary	Technology focus
---------------------	----------------	-------------------------

Architecture	Summary	Technology focus
Master data management with Azure and CluedIn	Use CluedIn eventual connectivity data integration to blend data from many siloed data sources and prepare it for analytics and business operations. CluedIn takes input from on-premises accessible systems like Oracle.	Databases
Migrate IBM mainframe apps to Azure with TmaxSoft OpenFrame	Migrate IBM zSeries mainframe applications to Azure. Use a no-code approach that TmaxSoft OpenFrame provides.	Mainframe
	OpenFrame can integrate with RDBMSs like Oracle.	
Oracle Database migration to Azure	Migrate an Oracle database and its applications to Azure. Use Oracle Active Data Guard for the database, and use Azure Load Balancer for the application tier.	Oracle
Oracle Database migration: Cross-cloud connectivity	Create a connection between your existing Oracle database and your Azure applications.	Oracle
Oracle Database migration: Lift and shift	Lift and shift your Oracle database from an Oracle environment to Azure Virtual Machines.	Oracle
Oracle Database migration: Refactor	Refactor your Oracle database by using Azure Database Migration Service, and move it to PostgreSQL.	Oracle
Oracle Database migration: Rearchitect	Rearchitect your Oracle database by using Azure SQL Managed Instance.	Oracle
Oracle Database with Azure NetApp Files	Implement a high-bandwidth, low-latency solution for Oracle Database workloads. Use Azure NetApp Files to get enterprise-scale performance and to reduce costs.	Storage
Overview of Oracle Database migration	Learn about Oracle Database migration paths and the methods you can use to migrate your schema to SQL or PostgreSQL.	Oracle
Refactor mainframe applications with Advanced	Learn how to use the automated COBOL refactoring solution from Advanced to modernize your mainframe COBOL applications, run them on Azure, and reduce costs. Use Oracle databases on VMs for persistent data.	Mainframe
Run Oracle databases on Azure	Use a canonical architecture to achieve high availability for Oracle Database Enterprise Edition on Azure.	Oracle

Architecture	Summary	Technology focus
Run SAP NetWeaver in Windows on Azure	Learn proven practices for running SAP NetWeaver in a Windows environment on Azure, with high availability. Oracle is one recommended database.	SAP
SAP deployment on Azure using an Oracle database	Learn proven practices for running SAP on Oracle in Azure, with high availability.	Oracle
Security considerations for highly sensitive IaaS apps in Azure	Learn about VM security, encryption, NSGs, perimeter networks (also known as DMZs), access control, and other security considerations for highly sensitive IaaS and hybrid apps. A common replication scenario for IaaS architectures uses Oracle Active Data Guard.	Security
SWIFT's Alliance Access with Alliance Connect Virtual on Azure	View a reference architecture for deploying and running SWIFT Alliance Access with Alliance Connect Virtual on Azure. An Alliance Access component contains an embedded Oracle database.	Networking
SWIFT's Alliance Messaging Hub (AMH) with Alliance Connect Virtual	Run SWIFT AMH on Azure. This messaging solution helps financial institutions securely and efficiently bring new services to market. A key component, the AMH node, runs an Oracle database.	Networking

Postman

Architecture	Summary	Technology focus
Design APIs for microservices	Learn about good API design in a microservices architecture. IDLs used to define APIs can be consumed by API testing tools like Postman.	Microservices
Gridwich local development environment setup	Set up a local development environment to work with Gridwich. Postman is an optional component in the configuration.	Media
Unified logging for microservices apps	Learn about logging, tracing, and monitoring for microservices apps.	Microservices

Profisee

Architecture	Summary	Technology focus
Data governance with Profisee and Azure Purview	Integrate Profisee master data management with Azure Purview to build a foundation for data governance and management.	Databases
Master data management with Profisee and Azure Data Factory	Integrate Profisee master data management with Data Factory to deliver high quality, trusted data for Azure Synapse and all analytics applications. Postman is recommended for synthetic logging.	Databases

Qlik

Architecture	Summary	Technology focus
Mainframe and midrange data replication to Azure using Qlik	Learn how Qlik Replication is a valuable tool for migrating mainframe and midrange systems to the cloud, or for extending such systems with cloud applications.	Mainframe

Raincode

Architecture	Summary	Technology focus
Rehost mainframe applications to Azure with Raincode compilers	Learn how the Raincode COBOL compiler modernizes mainframe legacy applications.	Mainframe

SAP

Architecture	Summary	Technology focus
Add a mobile front end to a legacy app	Learn about a solution that uses Azure SQL Database and SAP to consolidate data from multiple business systems and surface it through web and mobile front ends.	Mobile
Custom mobile workforce app	Learn about a mobile workforce app architecture that uses Active Directory to secure corporate data from an SAP backend system.	Mobile

Architecture	Summary	Technology focus
Development and test environments for SAP workloads on Azure	Learn how to establish non-production development and test environments for SAP NetWeaver in a Windows or Linux environment on Azure.	SAP
Master data management with Azure and CluedIn	Use CluedIn eventual connectivity data integration to blend data from many siloed data sources and prepare it for analytics and business operations. CluedIn takes input from on-premises accessible systems like SAP.	Databases
Multitier web application built for HA/DR	Learn how to create a resilient multitier web application built for high availability and disaster recovery on Azure. Common scenarios include any mission-critical application that runs on Windows or Linux, including applications like SAP.	Networking
Run SAP BW/4HANA with Linux VMs	Learn about the SAP BW/4HANA application tier and how it's suitable for a high availability small-scale production environment of SAP BW/4HANA on Azure.	SAP
Run SAP HANA for Linux VMs in scale-up systems	Learn proven practices for running SAP HANA in a high availability scale-up environment that supports disaster recovery.	SAP
Run SAP HANA Large Instances	Learn proven practices for running SAP HANA in a high availability environment on Azure Large Instances.	SAP
Run SAP NetWeaver in Windows on Azure	Learn proven practices for running SAP NetWeaver in a Windows environment on Azure, with high availability.	SAP
SAP deployment on Azure using an Oracle database	Learn proven practices for running SAP on Oracle in Azure, with high availability.	SAP
SAP on Azure architecture design	Review a set of guiding tenets to help ensure the quality of SAP workloads that run on Azure.	SAP
SAP NetWeaver on SQL Server	Build an SAP landscape on NetWeaver by using Azure Virtual Machines to host SAP applications and a SQL Server database.	SAP
SAP S/4HANA for Large Instances	With large SAP HANA instances, use Azure Virtual Machines, OS clustering, and NFS storage for scalability, performance, high reliability, and disaster recovery.	SAP

Architecture	Summary	Technology focus
SAP S/4HANA in Linux on Azure	Review proven practices for running SAP S/4HANA in a Linux environment on Azure, with high availability.	SAP
SAP workload automation using SUSE on Azure	Use this solution to bolster productivity and facilitate innovation.	SAP

SAS

Architecture	Summary	Technology focus
SAS on Azure architecture	Learn how to run SAS analytics products on Azure. See guidelines for designing and implementing cloud solutions for SAS Viya and SAS Grid.	Compute

Sitecore

Architecture	Summary	Technology focus
Scalable Sitecore marketing website	Learn how the Sitecore Experience Platform (XP) provides the data, integrated tools, and automation you need to engage customers throughout an iterative lifecycle.	Web

Skytap

Architecture	Summary	Technology focus
Migrate AIX workloads to Skytap on Azure	Learn how to migrate AIX logical partitions (LPARs) to Skytap on Azure.	Mainframe
Migrate IBM i series to Azure with Skytap	Learn how to use the native IBM i backup and recovery services with Azure components.	Mainframe

Software AG

Architecture	Summary	Technology focus
Refactor mainframe computer systems that run Adabas & Natural	Learn how to modernize mainframe computer systems that run Adabas & Natural and move them to the cloud.	Mainframe

Stromasys

Architecture	Summary	Technology focus
Stromasys Charon-SSP Solaris emulator on Azure VMs	Learn how the Charon-SSP cross-platform hypervisor emulates legacy Sun SPARC systems on industry standard x86-64 computer systems and VMs.	Mainframe

SWIFT

Architecture	Summary	Technology focus
SWIFT's Alliance Access with Alliance Connect Virtual on Azure	View a reference architecture for deploying and running SWIFT Alliance Access with Alliance Connect Virtual on Azure.	Networking
SWIFT Alliance Cloud on Azure	Deploy Azure infrastructure for SWIFT Alliance Cloud.	Networking
SWIFT Alliance Connect Virtual on Azure	View a series of articles about SWIFT Alliance Connect Virtual components that can be deployed on Azure.	Security
SWIFT Alliance Lite2 on Azure	Deploy SWIFT Alliance Lite2 on Azure. Migrate an existing deployment from on-premises or create a new deployment.	Networking
SWIFT's AMH with Alliance Connect Virtual	Run SWIFT AMH on Azure. Use this messaging solution with the Alliance Connect Virtual networking solution, which also runs on Azure.	Networking

Syncier

Architecture	Summary	Technology focus

Architecture	Summary	Technology focus
GitOps for Azure Kubernetes Service	View a GitOps solution for an AKS cluster. This solution provides full audit capabilities, policy enforcement, and early feedback. Syncier Security Tower provides an overview of all AKS clusters and helps manage policies.	Containers

TmaxSoft

Architecture	Summary	Technology focus
Migrate IBM mainframe apps to Azure with TmaxSoft OpenFrame	Migrate IBM zSeries mainframe applications to Azure. Use a no-code approach that TmaxSoft OpenFrame provides.	Mainframe

Unisys

Architecture	Summary	Technology focus
Unisys ClearPath Forward mainframe rehost to Azure using Unisys virtualization	Use virtualization technologies from Unisys and Azure to migrate from a Unisys ClearPath Forward Libra (legacy Burroughs A Series/MCP) mainframe.	Mainframe

Related resources

- [Apache open-source scenarios on Azure](#)
- [Open-source scenarios on Azure](#)
- [Scenarios featuring Microsoft on-premises technologies](#)
- [Architecture for startups](#)
- [Azure and Power Platform scenarios](#)
- [Azure and Microsoft 365 scenarios](#)
- [Azure and Dynamics 365 scenarios](#)
- [Azure for AWS professionals](#)
- [Azure for Google Cloud professionals](#)

Azure for AWS professionals

Article • 05/29/2023

This series of articles helps Amazon Web Services (AWS) experts understand the basics of Microsoft Azure accounts, platform, and services. These articles also cover key similarities and differences between AWS and Azure.

These articles describe:

- How Azure organizes accounts and resources.
- How Azure structures available solutions.
- How the major Azure services differ from AWS services.

To quickly find the comparable services across the different technology categories, see [AWS to Azure services comparison](#).

Similarities and differences

Like AWS, Microsoft Azure builds on a core set of compute, storage, database, and networking services. In many cases, the platforms offer similar products and services. For example, both AWS and Azure can use Linux distributions and open-source software (OSS) technologies. Both platforms support building highly available solutions on Windows or Linux hosts.

While the capabilities of both platforms are similar, the resources that provide those capabilities are often organized differently. Azure and AWS built their capabilities independently over time, so the platforms have important implementation and design differences. Exact one-to-one correspondences between the services that you need to build a solution aren't always clear.

Sometimes, only one of the platforms offers a particular service. For a complete listing and charts of how services map between the platforms, see [AWS to Azure services comparison](#).

Not all Azure products and services are available in all regions. For details, see [Products available by region](#). For Azure product and service uptime guarantees and downtime credit policies, see [Service-level agreements](#).

Compare Azure and AWS core components

Many Azure and AWS core components have similar functionality. The following articles compare the platforms' capabilities in these core areas:

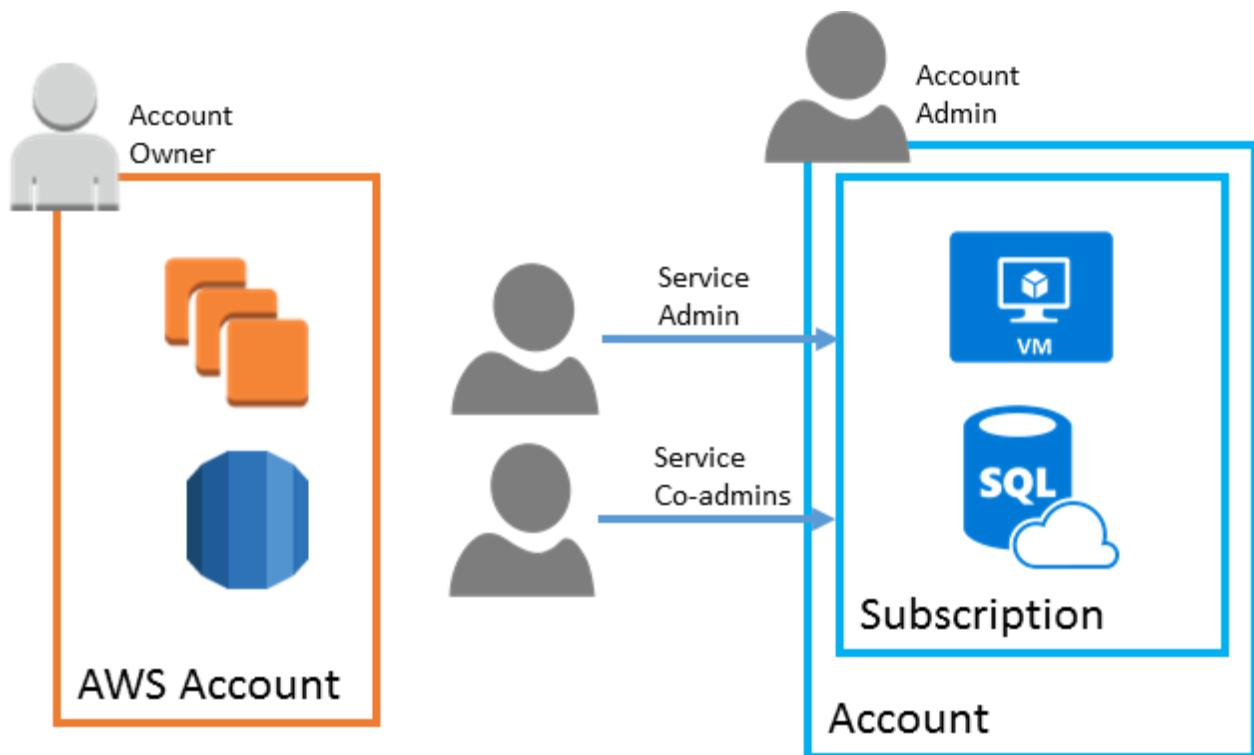
- [Azure and AWS accounts and subscriptions](#)
- [Compute services on Azure and AWS](#)
- [Relational database technologies on Azure and AWS](#)
- [Messaging services on Azure and AWS](#)
- [Networking on Azure and AWS](#)
- [Regions and zones on Azure and AWS](#)
- [Resource management on Azure and AWS](#)
- [Multi-cloud security and identity with Azure and AWS](#)
- [Compare storage on Azure and AWS](#)

Azure and AWS accounts and subscriptions

Article • 10/09/2023

Azure services can be purchased using several pricing options, depending on your organization's size and needs. See the [pricing overview](#) page for details.

Azure subscriptions are a grouping of resources with an assigned owner responsible for billing and permissions management. Unlike AWS, where any resources created under the AWS account are tied to that account, subscriptions exist independently of their owner accounts, and can be reassigned to new owners as needed.



Comparison of structure and ownership of AWS accounts and Azure subscriptions

An Azure account represents a billing relationship and Azure subscriptions help you organize access to Azure resources. Account Administrator, Service Administrator, and Co-Administrator are the three classic subscription administrator roles in Azure:

- **Account Administrator.** The subscription owner and the billing owner for the resources used in the subscription. The account administrator can only be changed by transferring ownership of the subscription. Only one Account administrator is assigned per Azure Account.

- **Service Administrator.** This user has rights to create and manage resources in the subscription, but is not responsible for billing. By default, for a new subscription, the Account Administrator is also the Service Administrator. The account administrator can assign a separate user to the service administrator for managing the technical and operational aspects of a subscription. Only one service administrator is assigned per subscription.
- **Co-administrator.** There can be multiple co-administrators assigned to a subscription. Co-administrators have the same access privileges as the Service Administrator, but they cannot change the service administrator.

Below the subscription level user roles and individual permissions can also be assigned to specific resources, similarly to how permissions are granted to IAM users and groups in AWS. In Azure, all user accounts are associated with either a Microsoft Account or Organizational Account (an account managed through a Microsoft Entra ID).

Like AWS accounts, subscriptions have default service quotas and limits. For a full list of these limits, see [Azure subscription and service limits, quotas, and constraints](#). These limits can be increased up to the maximum by [filing a support request in the management portal](#).

See also

- [Classic subscription administrator roles, Azure roles, and Microsoft Entra roles](#)
- [How to add or change Azure administrator roles](#)
- [How to download your Azure billing invoice and daily usage data](#)

Compute services on Azure and AWS

Article • 05/19/2023

This article compares the core compute services that Microsoft Azure and Amazon Web Services (AWS) offer.

- For links to articles that compare other AWS and Azure services, see [Azure for AWS professionals](#).
- For a complete listing and charts showing service mapping between AWS and Azure, see [AWS to Azure services comparison](#).
- [Browse Azure compute architectures](#).

Compare AWS and Azure compute services

The following tables describe and compare the core compute services on Amazon Web Services (AWS) and Azure.

Virtual machines and servers

Virtual machines (VMs) and servers allow users to deploy, manage, and maintain OS and other software. Users pay for what they use, with the flexibility to change sizes.

AWS service	Azure service	Description
Amazon EC2 Instance Types	Azure Virtual Machines	AWS and Azure on-demand VMs bill per seconds used. Although AWS instance types and Azure VM sizes have similar categories, the exact RAM, CPU, and storage capabilities differ. For information about Azure VM sizes, see Azure VM sizes .
VMware Cloud on AWS	Azure VMware Solution	AWS and Azure solutions let you move VMware vSphere-based workloads and environments to the cloud. Azure VMware Solution is a VMware-verified Microsoft service that runs on Azure infrastructure. You can manage existing environments with VMware solution tools, while modernizing applications with cloud native services.
AWS Parallel Cluster	Azure CycleCloud	Create, manage, operate, and optimize HPC and large compute clusters of any scale.

[View all the virtual machines architectures](#)

Autoscaling

Autoscaling lets you automatically change the number of VM instances. You set defined metrics and thresholds that determine when to add or remove instances.

AWS service	Azure service	Description
AWS Auto Scaling	Virtual machine scale sets, App Service autoscale	In Azure, virtual machine scale sets let you deploy and manage identical sets of VMs. The number of sets can autoscale. App Service autoscale lets you autoscale Azure App Service applications.

[View all the autoscaling architectures](#)

Batch processing

Batch processing runs large-scale parallel and high-performance computing applications efficiently in the cloud.

AWS service	Azure service	Description
AWS Batch	Azure Batch	Azure Batch helps you manage compute-intensive work across a scalable collection of VMs.

[View all the batch processing architectures](#)

Storage

Several services provide different types of data storage for VM disks.

AWS service	Azure service	Description
Disk volumes on Amazon Elastic Block Store (EBS)	Data disks in Azure Blob Storage	Data disks in blob storage provide durable data storage for Azure VMs. This storage is similar to AWS EC2 instance disk volumes on EBS.
Amazon EC2 instance store	Azure temporary storage	Azure temporary storage provides VMs with similar low-latency temporary read-write storage to EC2 instance storage, also called ephemeral storage.
Amazon EBS Provisioned IOPS Volume	Azure premium storage	Azure supports higher performance disk I/O with premium storage. This storage is similar to AWS Provisioned IOPS storage options.

AWS service	Azure service	Description
Amazon Elastic File System (EFS) 	Azure Files	Azure Files provides VMs with similar functionality to Amazon EFS.

[View all the storage architectures](#)

Containers and container orchestrators

Several AWS and Azure services provide containerized application deployment and orchestration.

AWS service	Azure service	Description
Amazon Elastic Container Service (Amazon ECS)  , AWS Fargate 	Azure Container Apps 	Azure Container Apps is a scalable service that lets you deploy thousands of containers without requiring access to the control plane.
Amazon Elastic Container Registry (Amazon ECR) 	Azure Container Registry 	Container registries store Docker formatted images and create all types of container deployments in the cloud.
Amazon Elastic Kubernetes Service (EKS) 	Azure Kubernetes Service (AKS) 	EKS and AKS let you orchestrate Docker containerized application deployments with Kubernetes. AKS simplifies monitoring and cluster management through auto upgrades and a built-in operations console. See Container runtime configuration for specifics on the hosting environment.
AWS App Mesh 	Open Service Mesh on AKS	The Open Service Mesh add-on integrates with features provided by Azure as well as open source projects.

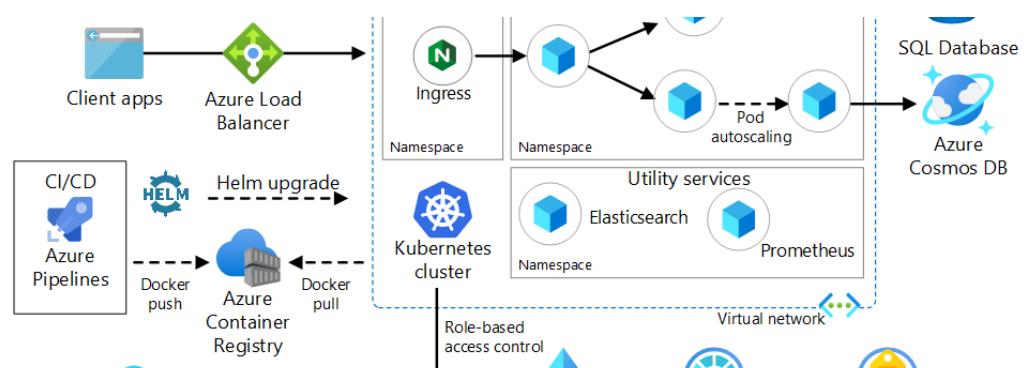
Example container architectures



Baseline architecture on Azure Kubernetes Service (AKS)

07/20/2020 37 min read

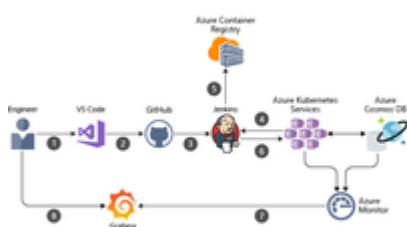
Deploy a baseline infrastructure that deploys an AKS cluster with focus on security.



Microservices architecture on Azure Kubernetes Service (AKS)

5/07/2020 17 min read

Deploy a microservices architecture on Azure Kubernetes Service (AKS)



CI/CD pipeline for container-based workloads

7/05/2018 7 min read

Build a DevOps pipeline for a Node.js web app with Jenkins, Azure Container Registry, Azure Kubernetes Service, Azure Cosmos DB, and Grafana.

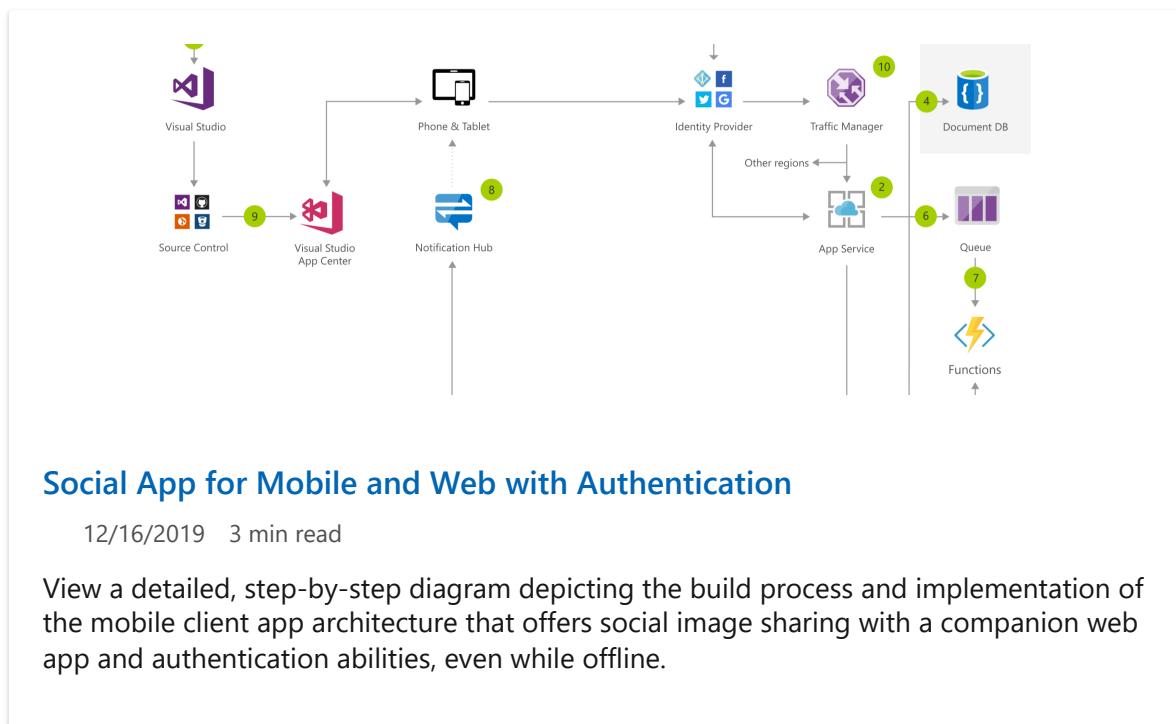
[View all the container architectures](#)

Serverless computing

Serverless computing lets you integrate systems and run backend processes without provisioning or managing servers.

AWS service	Azure service	Description
AWS Lambda	Azure Functions, WebJobs in Azure App Service	Azure Functions is the primary equivalent of AWS Lambda in providing serverless, on-demand code. AWS Lambda functionality also overlaps with Azure WebJobs, which let you schedule or continuously run background tasks.

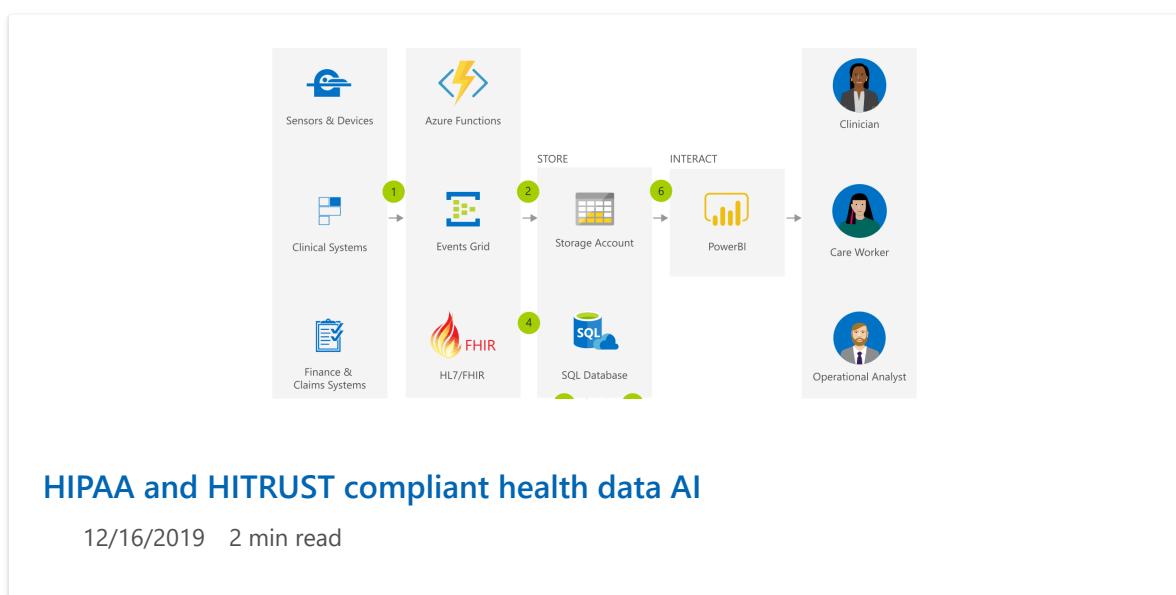
Example serverless architectures



Social App for Mobile and Web with Authentication

12/16/2019 3 min read

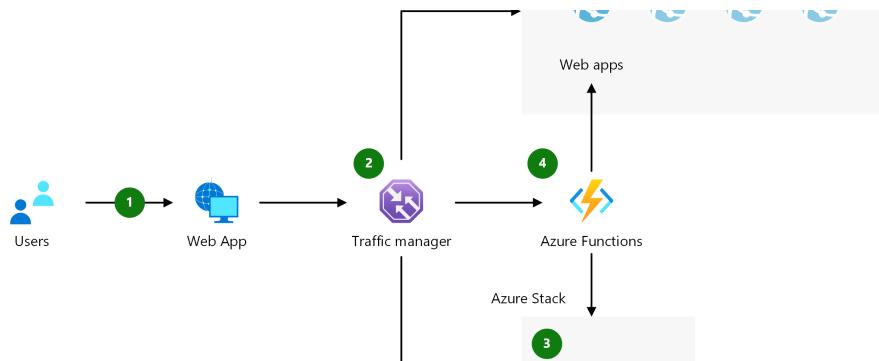
View a detailed, step-by-step diagram depicting the build process and implementation of the mobile client app architecture that offers social image sharing with a companion web app and authentication abilities, even while offline.



HIPAA and HITRUST compliant health data AI

12/16/2019 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.



Cross Cloud Scaling Architecture

12/16/2019 1 min read

Learn how to improve cross cloud scalability with solution architecture that includes Azure Stack. A step-by-step flowchart details instructions for implementation.

[View all the serverless architectures](#)

Contributors

This article is maintained by Microsoft. It was originally written by the following contributors.

Principal author:

- [Kobi Levi](#) | Cloud Solution Architect

Next steps

- Quickstart: Create a Linux virtual machine in the Azure portal
- Create a Node.js web app in Azure
- Getting started with Azure Functions
- Azure Kubernetes Service (AKS) architecture design

Related resources

- Baseline architecture for an Azure Kubernetes Service (AKS) cluster
- Microservices architecture on Azure Kubernetes Service
- CI/CD pipeline for container-based workloads
- Cross-cloud scaling with Azure Functions

- Run a Linux VM on Azure
- Basic web application
- Baseline App Service web application with zone redundancy
- Active-passive multi-region App Service web application
- Social app for mobile and web with authentication

Relational database technologies on Azure and AWS

Article • 11/15/2022

RDS and Azure relational database services

Azure provides several different relational database services that are the equivalent of AWS' Relational Database Service (RDS). These include:

- [SQL Database](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)
- [Azure Database for MariaDB](#)

Other database engines such as [SQL Server ↗](#), [Oracle ↗](#), and [MySQL](#) can be deployed using Azure VM Instances.

Costs for AWS RDS are determined by the amount of hardware resources that your instance uses, like CPU, RAM, storage, and network bandwidth. In the Azure database services, cost depends on your database size, concurrent connections, and throughput levels.

See also

- [Azure SQL Database Tutorials](#)
- [Azure SQL Managed Instance](#)
- [Configure geo-replication for Azure SQL Database with the Azure portal](#)
- [Introduction to Azure Cosmos DB: A NoSQL JSON Database](#)
- [How to use Azure Table storage from Node.js](#)

Analytics and big data

Azure provides a package of products and services designed to capture, organize, analyze, and visualize large amounts of data consisting of the following services:

- [HDInsight](#): managed Apache distribution that includes Hadoop, Spark, Storm, or HBase.

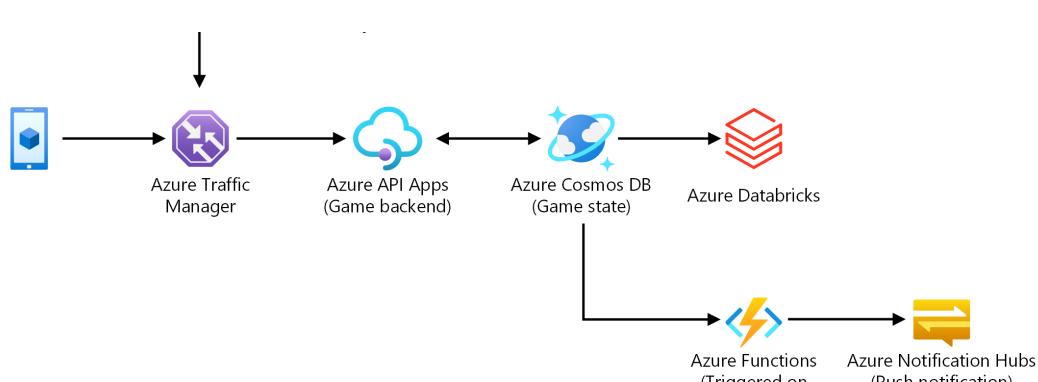
- [Data Factory](#): provides data orchestration and data pipeline functionality.
- [Azure Synapse Analytics](#): an enterprise analytics service that accelerates time to insight, across data warehouses and big data systems.
- [Azure Databricks](#): a unified analytics platform for data analysts, data engineers, data scientists, and machine learning engineers.
- [Data Lake Store](#): analytics service that brings together enterprise data warehousing and big data analytics. Query data on your terms, using either serverless or dedicated resources—at scale.
- [Machine Learning](#): used to build and apply predictive analytics on data.
- [Stream Analytics](#): real-time data analysis.
- [Data Lake Analytics](#): large-scale analytics service optimized to work with Data Lake Store
- [Power BI ↗](#): a business analytics service that provides the capabilities to create rich interactive data visualizations.

Service comparison

Type	AWS Service	Azure Service	Description
Relational database	RDS ↗	SQL Database ↗ Database for MySQL ↗ Database for PostgreSQL ↗ Database for MariaDB ↗	Managed relational database services in which resiliency, scale and maintenance are primarily handled by the Azure platform.

Type	AWS Service	Azure Service	Description
Serverless relational database	Amazon Aurora Serverless ↗ Amazon SimpleDB ↗ Amazon DocumentDB ↗ (Document) Amazon Neptune ↗ (Graph)	Azure SQL Database serverless Serverless SQL pool in Azure Synapse Analytics	Database offerings that automatically scales compute based on the workload demand. You're billed per second for the actual compute used (Azure SQL)/data that's processed by your queries (Azure Synapse Analytics Serverless).
NoSQL	DynamoDB ↗ Amazon DocumentDB ↗ (Document)	Azure Cosmos DB ↗ Cache for Redis ↗	Azure Cosmos DB is a globally distributed, multi-model database that natively supports multiple data models including key-value pairs, documents, graphs, and columnar.
Caching	ElastiCache ↗ Amazon MemoryDB for Redis ↗	Cache for Redis ↗	An in-memory-based, distributed caching service that provides a high-performance store that's typically used to offload nontransactional work from a database.
Database migration	Database Migration Service ↗	Database Migration Service ↗	A service that executes the migration of database schema and data from one database format to a specific database technology in the cloud.

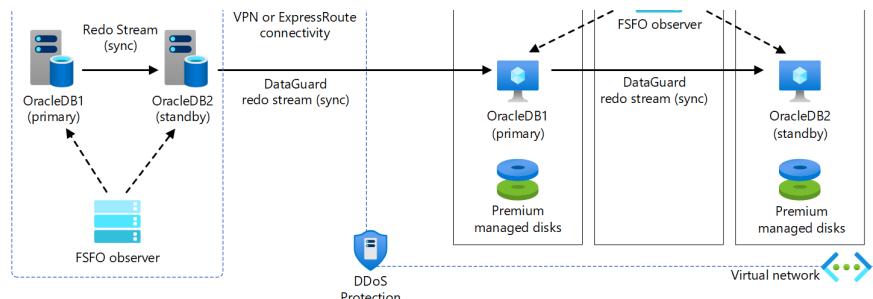
Database architectures



Gaming using Azure Cosmos DB

12/16/2019 1 min read

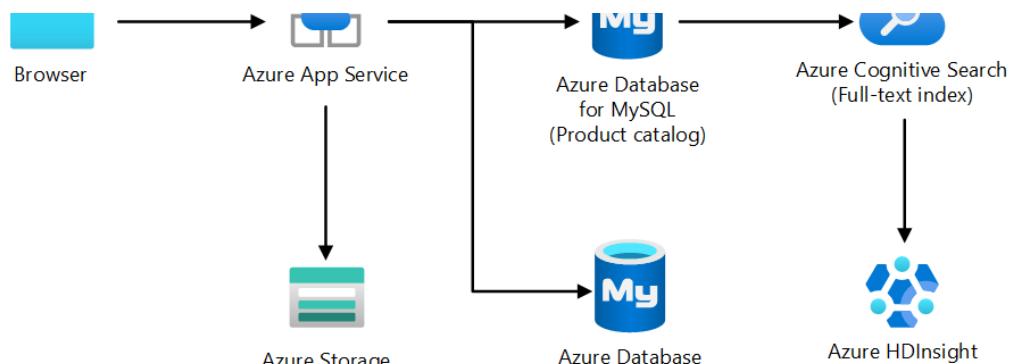
Elastically scale your database to accommodate unpredictable bursts of traffic and deliver low-latency multi-player experiences on a global scale.



Oracle Database Migration to Azure

12/16/2019 2 min read

Oracle DB migrations can be accomplished in multiple ways. This architecture covers one of these options wherein Oracle Active Data Guard is used to migrate the Database.



Retail and e-commerce using Azure MySQL

12/16/2019 1 min read

Build secure and scalable e-commerce solutions that meet the demands of both customers and business using Azure Database for MySQL.

[view all](#)

See also

- [Azure AI Gallery ↗](#)
- [Cloud-scale analytics ↗](#)

- Big data architecture style
- Azure Data Lake and Azure HDInsight blog

Messaging services on Azure and AWS

Article • 10/23/2023

Simple Email Service

AWS provides the Simple Email Service (SES) for sending notification, transactional, or marketing emails. In Azure, you can send emails with [Azure Communication Services](#) or third-party solutions, like [SendGrid](#). Both of these options provide email services that can be incorporated into solutions to cater for various use cases.

Simple Queueing Service

AWS Simple Queueing Service (SQS) provides a messaging system for connecting applications, services, and devices within the AWS platform. Azure has two services that provide similar functionality:

- [Queue storage](#) is a cloud messaging service that allows communication between application components within Azure.
- [Service Bus](#) is a robust messaging system for connecting applications, services, and devices. By using the related [Service Bus relay](#), Service Bus can also connect to remotely hosted applications and services.

Integrating between Azure and AWS messaging services

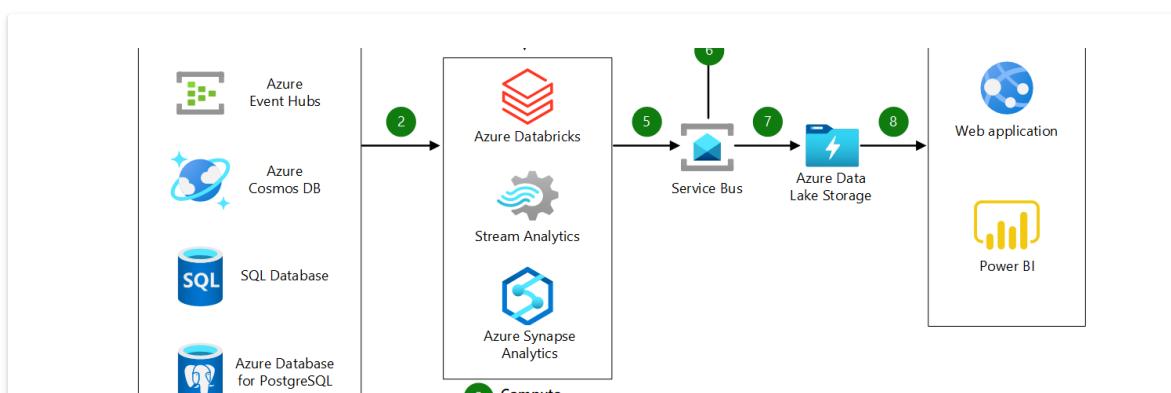
If there is one set of components using Amazon SQS that needs to integrate with another set of components that uses Azure Service Bus, or vice versa, that can be done using the [Messaging Bridge pattern](#).

Messaging components

AWS service	Azure service	Description
Simple Queue Service (SQS)	Queue Storage	Provides a managed message queueing service for communicating between decoupled application components.
Simple Notification Service (SNS)	Service Bus	Supports a set of cloud-based, message-oriented middleware technologies, including reliable message queuing and durable publish/subscribe messaging.

AWS service	Azure service	Description
Amazon EventBridge ↗	Event Grid ↗	A fully managed event routing service that allows for uniform event consumption using a publish/subscribe model.
Amazon Kinesis ↗	Event Hubs ↗	A fully managed, real-time data ingestion service. Stream millions of events per second, from any source, to build dynamic data pipelines and to immediately respond to business challenges.
Amazon MQ ↗	Service Bus	Service Bus Premium is fully compliant with the Java/Jakarta EE Java Message Service (JMS) 2.0 API. Service Bus Standard supports the JMS 1.1 subset focused on queues.

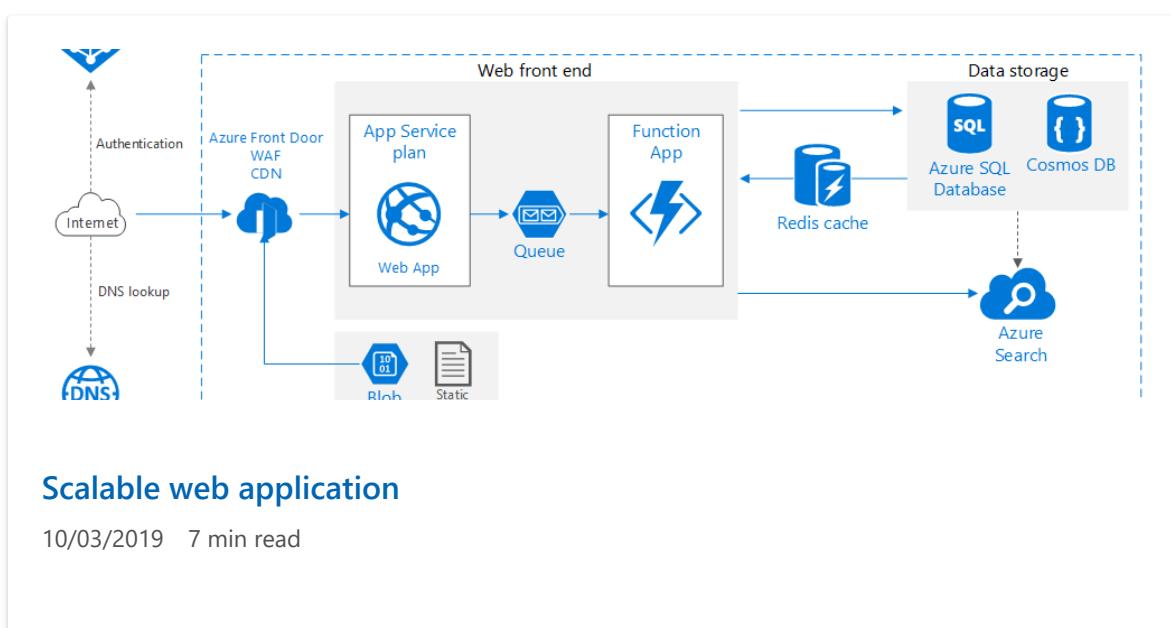
Messaging architectures



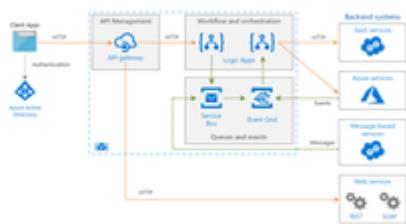
Anomaly Detector Process

12/16/2019 1 min read

Learn more about Anomaly Detector with a step-by-step flowchart that details the process. See how anomaly detection models are selected with time-series data.



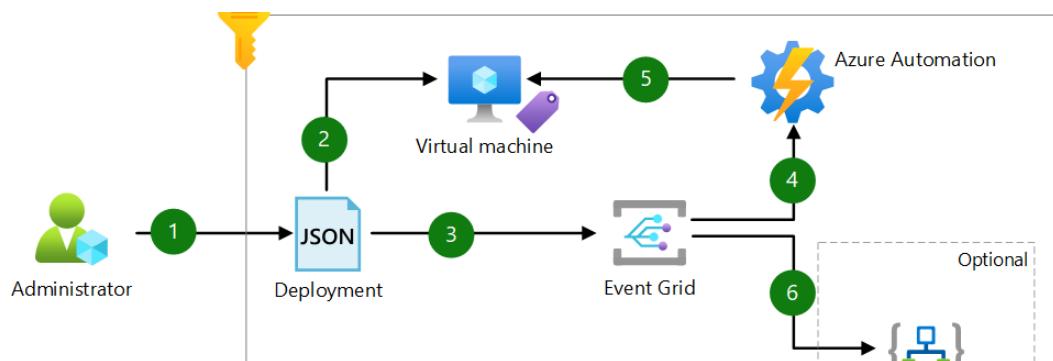
Use the proven practices in this reference architecture to improve scalability and performance in an Azure App Service web application..



Enterprise integration using queues and events

12/03/2018 5 min read

Recommended architecture for implementing an enterprise integration pattern with Azure Logic Apps, Azure API Management, Azure Service Bus, and Azure Event Grid.



Ops automation using Event Grid

12/16/2019 1 min read

Event Grid allows you to speed automation and simplify policy enforcement. For example, Event Grid can notify Azure Automation when a virtual machine is created, or a SQL Database is spun up. These events can be used to automatically check that service configurations are compliant, put metadata into operations tools, tag virtual machines, ...

Networking on Azure and AWS

Article • 11/15/2022

Elastic Load Balancing, Azure Load Balancer, and Azure Application Gateway

The Azure equivalent of the Elastic Load Balancing services are:

- [Load Balancer](#): Provides the same network layer 4 capabilities as the AWS Network Load Balancer and Classic Load Balancer, allowing you to distribute traffic for multiple VMs at the network level. It also provides a failover capability.
- [Application Gateway](#): Offers application-level rule-based routing comparable to the AWS Application Load Balancer.

Route 53, Azure DNS, and Azure Traffic Manager

In AWS, Route 53 provides both DNS name management and DNS-level traffic routing and failover services. In Azure this is handled through two services:

- [Azure DNS](#) provides domain and DNS management.
- [Traffic Manager](#) provides DNS level traffic routing, load balancing, and failover capabilities.

Direct connect and Azure ExpressRoute

Azure provides similar site-to-site dedicated connections through its [ExpressRoute](#) service. ExpressRoute allows you to connect your local network directly to Azure resources using a dedicated private network connection. Azure also offers more conventional [site-to-site VPN connections](#) at a lower cost.

Route tables

AWS provides route tables that contain routes to direct traffic, from a subnet/gateway subnet to the destination. In Azure, this feature is called user-defined routes.

With [user-defined routes](#), you can create custom or user-defined (static) routes in Azure, to override Azure's default system routes, or to add more routes to a subnet's route table.

Private Link

Similar to AWS PrivateLink, [Azure Private Link](#) provides private connectivity from a virtual network to an Azure platform as a service (PaaS) solution, a customer-owned service, or a Microsoft partner service.

VPC Peering, Azure VNet Peering

In AWS, a VPC peering connection is a networking connection between two VPCs, which enables you to route traffic between them using private IPv4 addresses or IPv6 addresses.

[Azure virtual network \(VNet\) peering](#) enables you to seamlessly connect two or more Virtual Networks in Azure. The virtual networks appear as one for connectivity purposes. The traffic between virtual machines in peered virtual networks uses the Microsoft backbone infrastructure. Like traffic between virtual machines in the same network, traffic is routed through Microsoft's private network only.

Content delivery networks - CloudFront and Azure Front Door

In AWS, CloudFront provides CDN services, to globally deliver data, videos, applications, and APIs. This is similar to Azure Front Door.

[Azure Front Door](#) is a modern cloud content delivery network (CDN) service that delivers high performance, scalability, and secure user experiences for your content and applications. For a full list of Azure Front Door product offerings, see [Overview of Azure Front Door tiers](#).

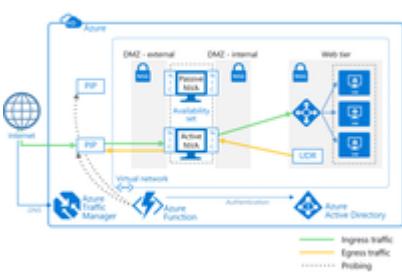
Network service comparison

Area	AWS service	Azure service	Description
------	-------------	---------------	-------------

Area	AWS service	Azure service	Description
Cloud virtual networking	Virtual Private Cloud (VPC) ↗	Virtual Network ↗	Provides an isolated, private environment in the cloud. Users have control over their virtual networking environment, including selection of their own IP address range, creation of subnets, and configuration of route tables and network gateways.
NAT gateways	NAT Gateways ↗	Virtual Network NAT	A service that simplifies outbound-only Internet connectivity for virtual networks. When configured on a subnet, all outbound connectivity uses your specified static public IP addresses. Outbound connectivity is possible without a load balancer or public IP addresses directly attached to virtual machines.
Cross-premises connectivity	VPN Gateway ↗	VPN Gateway	Connects Azure virtual networks to other Azure virtual networks, or customer on-premises networks (Site To Site). Allows end users to connect to Azure services through VPN tunneling (Point To Site).
DNS management	Route 53 ↗	DNS ↗	Manage your DNS records using the same credentials and billing and support contract as your other Azure services
DNS-based routing	Route 53 ↗	Traffic Manager ↗	A service that hosts domain names, plus routes users to Internet applications, connects user requests to datacenters, manages traffic to apps, and improves app availability with automatic failover.
Dedicated network	Direct Connect ↗	ExpressRoute ↗	Establishes a dedicated, private network connection from a location to the cloud provider (not over the Internet).
Load balancing	Network Load Balancer ↗	Load Balancer ↗	Azure Load Balancer load balances traffic at layer 4 (TCP or UDP). Standard Load Balancer also supports cross-region or global load balancing.
Application-level load balancing	Application Load Balancer ↗	Application Gateway ↗	Application Gateway is a layer 7 load balancer. It supports SSL termination, cookie-based session affinity, and round robin for load-balancing traffic.
Route table	Custom Route Tables ↗	User Defined Routes	Custom, or user-defined (static) routes to override default system routes, or to add more routes to a subnet's route table.

Area	AWS service	Azure service	Description
Private link	PrivateLink ↗	Azure Private Link ↗	Azure Private Link provides private access to services that are hosted on the Azure platform. This keeps your data on the Microsoft network.
Private PaaS connectivity	VPC endpoints ↗	Private Endpoint	Private Endpoint provides secured, private connectivity to various Azure platform as a service (PaaS) resources, over a backbone Microsoft private network.
Virtual network peering	VPC Peering ↗	VNET Peering ↗	VNet peering is a mechanism that connects two virtual networks (VNets) in the same region through the Azure backbone network. Once peered, the two virtual networks appear as one for all connectivity purposes.
Content delivery networks	CloudFront ↗	Front Door ↗	Azure Front Door is a modern cloud content delivery network (CDN) service that delivers high performance, scalability, and secure user experiences for your content and applications.
Network Monitoring	VPC Flow Logs ↗	Azure Network Watcher	Azure Network Watcher allows you to monitor, diagnose, and analyze the traffic in Azure Virtual Network.

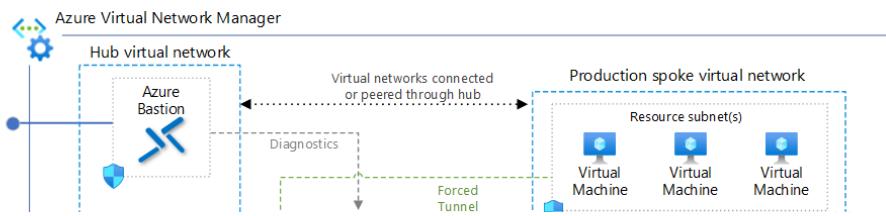
Networking architectures



Deploy highly available NVAs

12/08/2018 7 min read

Learn how to deploy network virtual appliances for high availability in Azure. This article includes example architectures for ingress, egress, and both.



[Hub-spoke network topology in Azure](#)

9/30/2020 7 min read

Learn how to implement a hub-spoke topology in Azure, where the hub is a virtual network and the spokes are virtual networks that peer with the hub.



[Implement a secure hybrid network](#)

1/07/2020 9 min read

See a secure hybrid network that extends an on-premises network to Azure with a perimeter network between the on-premises network and an Azure virtual network.

[view all](#)

See also

- [Create a virtual network using the Azure portal](#)
- [Plan and design Azure Virtual Networks](#)
- [Azure Network Security Best Practices](#)

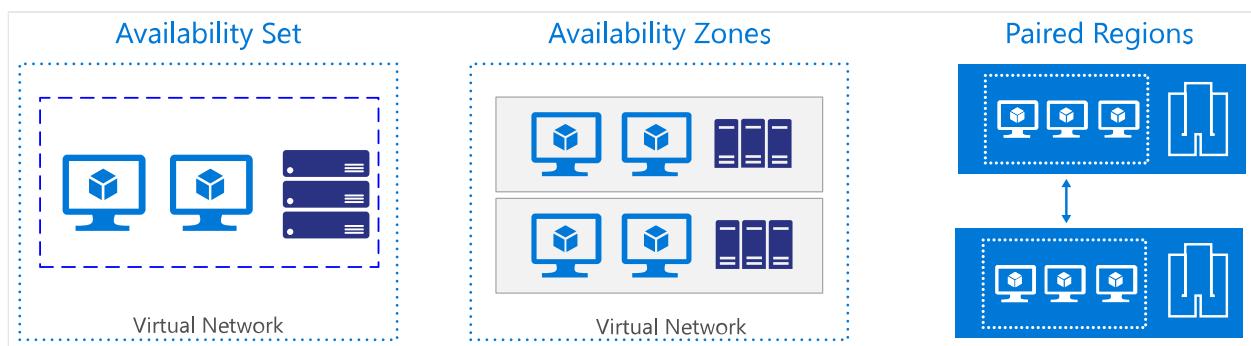
Regions and zones on Azure

Article • 03/29/2023

Failures can vary in the scope of their impact. Some hardware failures, such as a failed disk, may affect a single host machine. A failed network switch could affect a whole server rack. Less common are failures that disrupt a whole datacenter, such as loss of power in a datacenter. Rarely, an entire region could become unavailable.

One of the main ways to make an application resilient is through redundancy. But you need to plan for this redundancy when you design the application. Also, the level of redundancy that you need depends on your business requirements—not every application needs redundancy across regions to guard against a regional outage. In general, a tradeoff exists between greater redundancy and reliability versus higher cost and complexity.

In Azure, a region is divided into two or more Availability Zones. An Availability Zone corresponds with a physically isolated datacenter in the geographic region. Azure has numerous features for providing application redundancy at every level of potential failure, including **availability sets**, **availability zones**, and **paired regions**.



The following table summarizes each option.

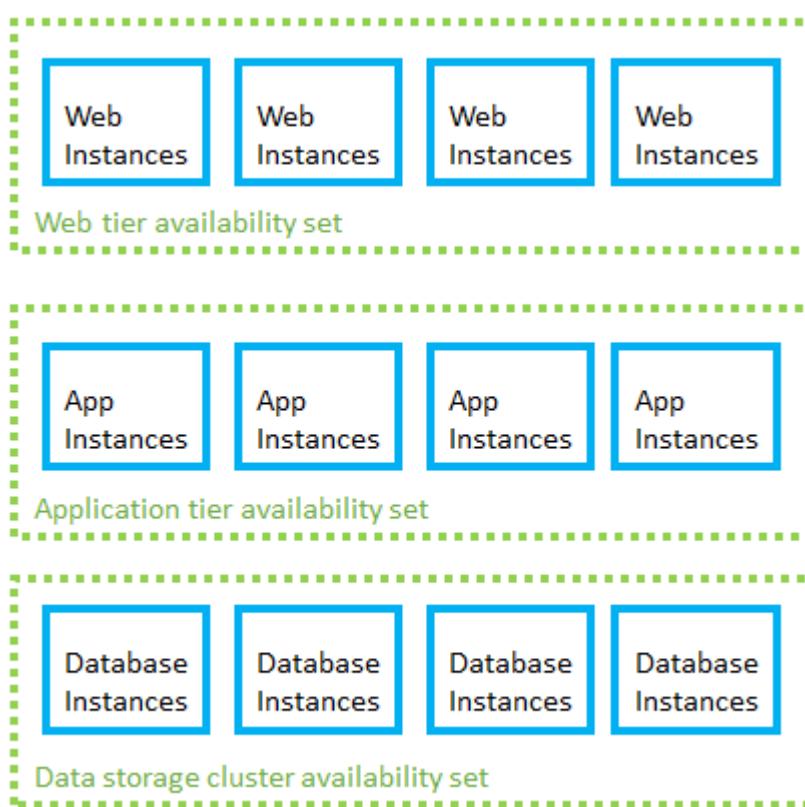
	Availability Set	Availability Zone	Paired region
Scope of failure	Rack	Datacenter	Region
Request routing	Load Balancer	Cross-zone Load Balancer	Traffic Manager
Network latency	Very low	Low	Mid to high
Virtual networking	VNet	VNet	Cross-region VNet peering

Availability sets

To protect against localized hardware failures, such as a disk or network switch failing, deploy two or more VMs in an availability set. An availability set consists of two or more *fault domains* that share a common power source and network switch. VMs in an availability set are distributed across the fault domains, so if a hardware failure affects one fault domain, network traffic can still be routed to the VMs in the other fault domains. For more information about Availability Sets, see [Manage the availability of Windows virtual machines in Azure](#).

When VM instances are added to availability sets, they are also assigned an [update domain](#). An update domain is a group of VMs that are set for planned maintenance events at the same time. Distributing VMs across multiple update domains ensures that planned update and patching events affect only a subset of these VMs at any given time.

Availability sets should be organized by the instance's role in your application to ensure one instance in each role is operational. For example, in a three-tier web application, create separate availability sets for the front-end, application, and data tiers.



Availability zones

An [Availability Zone](#) is a physically separate zone within an Azure region. Each Availability Zone has a distinct power source, network, and cooling. Deploying VMs across availability zones helps to protect an application against datacenter-wide failures.

Paired regions

To protect an application against a regional outage, you can deploy the application across multiple regions, using [Azure Traffic Manager](#) to distribute internet traffic to the different regions. Each Azure region is paired with another region. Together, these form a [regional pair](#). With the exception of Brazil South, regional pairs are located within the same geography in order to meet data residency requirements for tax and law enforcement jurisdiction purposes.

Unlike Availability Zones, which are physically separate datacenters but may be in relatively nearby geographic areas, paired regions are typically separated by at least 300 miles. This design ensures that large-scale disasters only affect one of the regions in the pair. Neighboring pairs can be set to sync database and storage service data, and are configured so that platform updates are rolled out to only one region in the pair at a time.

Azure [geo-redundant storage](#) is automatically backed up to the appropriate paired region. For all other resources, creating a fully redundant solution using paired regions means creating a full copy of your solution in both regions.

See also

- [Regions for virtual machines in Azure](#)
- [Availability options for virtual machines in Azure](#)
- [High availability for Azure applications](#)
- [Failure and disaster recovery for Azure applications](#)
- [Planned maintenance for Linux virtual machines in Azure](#)

Resource management on Azure and AWS

Article • 12/16/2022

The term "resource" in Azure is used in the same way as in AWS, meaning any compute instance, storage object, networking device, or other entity you can create or configure within the platform.

Azure resources are deployed and managed using one of two models: [Azure Resource Manager](#), or the older Azure [classic deployment model](#). Any new resources are created using the Resource Manager model.

Resource groups

Both Azure and AWS have entities called "resource groups" that organize resources such as VMs, storage, and virtual networking devices. However, [Azure resource groups](#) are not directly comparable to AWS resource groups.

While AWS allows a resource to be tagged into multiple resource groups, an Azure resource is always associated with one resource group. A resource created in one resource group can be moved to another group, but can only be in one resource group at a time. Resource groups are the fundamental grouping used by Azure Resource Manager.

Resources can also be organized using [tags](#). Tags are key-value pairs that allow you to group resources across your subscription irrespective of resource group membership.

Management interfaces

Azure offers several ways to manage your resources:

- [Web interface](#). Like the AWS Dashboard, the Azure portal provides a full web-based management interface for Azure resources.
- [REST API](#). The Azure Resource Manager REST API provides programmatic access to most of the features available in the Azure portal.
- [Command Line](#). The Azure CLI provides a command-line interface capable of creating and managing Azure resources. The Azure CLI is available for [Windows](#), [Linux](#), and [Mac OS](#).

- [PowerShell](#). The Azure modules for PowerShell allow you to execute automated management tasks using a script. PowerShell is available for [Windows](#), [Linux](#), and [Mac OS](#).
- [Templates](#). Azure Resource Manager templates provide similar JSON template-based resource management capabilities to the AWS CloudFormation service.

In each of these interfaces, the resource group is central to how Azure resources get created, deployed, or modified. This is similar to the role a "stack" plays in grouping AWS resources during CloudFormation deployments.

The syntax and structure of these interfaces are different from their AWS equivalents, but they provide comparable capabilities. In addition, many third-party management tools used on AWS, like [Hashicorp's Terraform](#) and [Netflix Spinnaker](#), are also available on Azure.

See also

- [Azure resource group guidelines](#)

Multicloud security and identity with Azure and Amazon Web Services (AWS)

Article • 10/09/2023

Many organizations are finding themselves with a de facto multicloud strategy, even if that wasn't their deliberate strategic intention. In a multicloud environment, it's critical to ensure consistent security and identity experiences to avoid increased friction for developers, business initiatives and increased organizational risk from cyberattacks taking advantage of security gaps.

Driving security and identity consistency across clouds should include:

- Multicloud identity integration
- Strong authentication and explicit trust validation
- Cloud Platform Security (multicloud)
- Microsoft Defender for Cloud
- Privilege Identity Management (Azure)
- Consistent end-to-end identity management

Multicloud identity integration

Customers using both Azure and AWS cloud platforms benefit from consolidating identity services between these two clouds using [Microsoft Entra ID](#) and Single Sign-on (SSO) services. This model allows for a consolidated identity plane through which access to services in both clouds can be consistently accessed and governed.

This approach allows for the rich role-based access controls in Microsoft Entra ID to be enabled across the Identity & Access Management (IAM) services in AWS using rules to associate the user.userprincipalname and user.assignrole attributes from Microsoft Entra ID into IAM permissions. This approach reduces the number of unique identities users and administrators are required to maintain across both clouds including a consolidation of the identity per account design that AWS employs. The [AWS IAM solution](#) allows for and specifically identifies Microsoft Entra ID as a federation and authentication source for their customers.

A complete walk-through of this integration can be found in the [Tutorial: Microsoft Entra single sign-on \(SSO\) integration with Amazon Web Services \(AWS\)](#).

Strong authentication and explicit trust validation

Because many customers continue to support a hybrid identity model for Active Directory services, it's increasingly important for security engineering teams to implement strong authentication solutions and block legacy authentication methods associated primarily with on-premises and legacy Microsoft technologies.

A combination of multi-factor authentication (MFA) and conditional access (CA) policies enable enhanced security for common authentication scenarios for end users in your organization. While MFA itself provides an increase level of security to confirm authentications, additional controls can be applied using [CA controls to block legacy authentication](#) to both Azure and AWS cloud environments. Strong authentication using only modern authentication clients is only possible with the combination of MFA and CA policies.

Cloud Platform Security (multicloud)

Once a common identity has been established in your multicloud environment, the [Cloud Platform Security \(CPS\)](#) service of [Microsoft Defender for Cloud Apps](#) can be used to discover, monitor, assess, and protect those services. Using the Cloud Discovery dashboard, security operations personnel can review the apps and resources being used across AWS and Azure cloud platforms. Once services are reviewed and sanctioned for use, the services can then be managed as enterprise applications in Microsoft Entra ID to enable SAML, password-based, and linked Single Sign-On mode for the convenience of users.

CPS also provides for the ability to assess the cloud platforms connected for misconfigurations and compliance using vendor specific recommended security and configuration controls. This design enables organizations to maintain a single consolidated view of all cloud platform services and their compliance status.

CPS also provides access and session control policies to prevent and protect your environment from risky endpoints or users when data exfiltration or malicious files are introduced into those platforms.

Microsoft Defender for Cloud

[Microsoft Defender for Cloud](#) provides unified security management and threat protection across your hybrid and multicloud workloads, including workloads in Azure,

Amazon Web Services (AWS), and Google Cloud Platform (GCP). Defender for Cloud helps you find and fix security vulnerabilities, apply access and application controls to block malicious activity, detect threats using analytics and intelligence, and respond quickly when under attack.

To [protect your AWS-based resources on Microsoft Defender for Cloud](#), you can connect an account with either the Classic cloud connectors experience or the Environment settings page (in preview), which is recommended.

Privileged Identity Management (Azure)

To limit and control access for your highest privileged accounts in Microsoft Entra ID, [Privileged Identity Management \(PIM\)](#) can be enabled to provide just-in-time access to services for Azure cloud services. Once deployed, PIM can be used to control and limit access using the assignment model for roles, eliminate persistent access for these privileged accounts, and provide additional discover and monitoring of users with these account types.

When combined with [Microsoft Sentinel](#), workbooks and playbooks can be established to monitor and raise alerts to your security operations center personnel when there is lateral movement of accounts that have been compromised.

Consistent end-to-end identity management

Ensure that all processes include an end-to-end view of all clouds as well as on-premises systems and that security and identity personnel are trained on these processes.

Using a single identity across Microsoft Entra ID, AWS Accounts and on-premises services enable this end-to-end strategy and allows for greater security and protection of accounts for privileged and non-privileged accounts. Customers who are currently looking to reduce the burden of maintaining multiple identities in their multicloud strategy adopt Microsoft Entra ID to provide consistent and strong control, auditing, and detection of anomalies and abuse of identities in their environment.

Continued growth of new capabilities across the Microsoft Entra ecosystem helps you stay ahead of threats to your environment as a result of using identities as a common control plane in your multicloud environments.

Next steps

- [Microsoft Entra B2B](#): enables access to your corporate applications from partner-managed identities.
- [Azure Active Directory B2C](#): service offering support for single sign-on and user management for consumer-facing applications.
- [Microsoft Entra Domain Services](#): hosted domain controller service, allowing Active Directory compatible domain join and user management functionality.
- [Getting started with Microsoft Azure security](#)
- [Azure Identity Management and access control security best practices](#)

Compare storage on Azure and AWS

Article • 07/27/2023

S3/EBS/EFS and Azure Storage

In the AWS platform, cloud storage is primarily broken down into three services:

- **Simple Storage Service (S3)**. Basic object storage that makes data available through an Internet accessible API.
- **Elastic Block Storage (EBS)**. Block level storage intended for access by a single VM.
- **Elastic File System (EFS)**. File storage meant for use as shared storage for up to thousands of EC2 instances.

In Azure Storage, subscription-bound [storage accounts](#) allow you to create and manage the following storage services:

- [Blob storage](#) stores any type of text or binary data, such as a document, media file, or application installer. You can set Blob storage for private access or share contents publicly to the Internet. Blob storage serves the same purpose as both AWS S3 and EBS.
- [Table storage](#) stores structured datasets. Table storage is a NoSQL key-attribute data store that allows for rapid development and fast access to large quantities of data. Similar to AWS' SimpleDB and DynamoDB services.
- [Queue storage](#) provides messaging for workflow processing and for communication between components of cloud services.
- [File storage](#) offers shared storage for legacy applications using the standard server message block (SMB) protocol. File storage is used in a similar manner to EFS in the AWS platform.

Glacier and Azure Storage

[Azure Archive Blob Storage](#) is comparable to AWS Glacier storage service. It is intended for rarely accessed data that is stored for at least 180 days and can tolerate several hours of retrieval latency.

For data that is infrequently accessed but must be available immediately when accessed, [Azure Cool Blob Storage tier](#) provides cheaper storage than standard blob storage. This

storage tier is comparable to AWS S3 - Infrequent Access storage service.

Storage comparison

Object storage

AWS service	Azure service	Description
Simple Storage Services (S3)	Blob storage	Object storage service, for use cases including cloud applications, content distribution, backup, archiving, disaster recovery, and big data analytics.

Virtual server disks

AWS service	Azure service	Description
Elastic Block Store (EBS)	managed disks	SSD storage optimized for I/O intensive read/write operations. For use as high-performance Azure virtual machine storage.

Shared files

AWS service	Azure service	Description
Elastic File System	Files	Provides a simple interface to create and configure file systems quickly, and share common files. Can be used with traditional protocols that access files over a network.

Archiving and backup

AWS service	Azure service	Description
S3 Infrequent Access (IA)	Storage cool tier	Cool storage is a lower-cost tier for storing data that is infrequently accessed and long-lived.
S3 Glacier , Deep Archive	Storage archive access tier	Archive storage has the lowest storage cost and higher data retrieval costs compared to hot and cool storage.
Backup	Backup	Back up and recover files and folders from the cloud, and provide offsite protection against data loss.

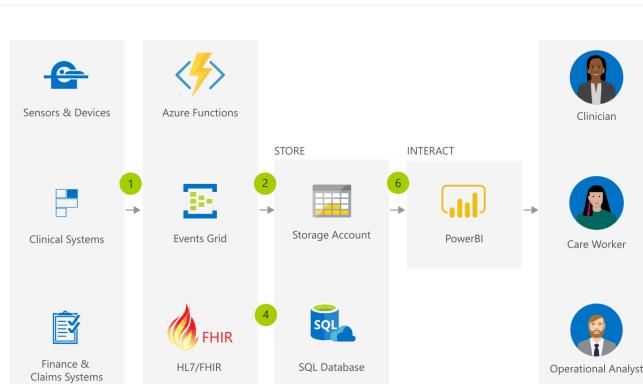
Hybrid storage

AWS service	Azure service	Description
Storage Gateway	StorSimple	Integrates on-premises IT environments with cloud storage. Automates data management and storage, plus supports disaster recovery.
DataSync	File Sync	Azure Files can be deployed in two main ways: by directly mounting the serverless Azure file shares or by caching Azure file shares on-premises using Azure File Sync.

Bulk data transfer

AWS service	Azure service	Description
Import/Export Disk	Import/Export	A data transport solution that uses secure disks and appliances to transfer large amounts of data. Also offers data protection during transit.
Import/Export Snowball , Snowball Edge , Snowmobile	Data Box	Petabyte- to exabyte-scale data transport solution that uses secure data storage devices to transfer large amounts of data to and from Azure.

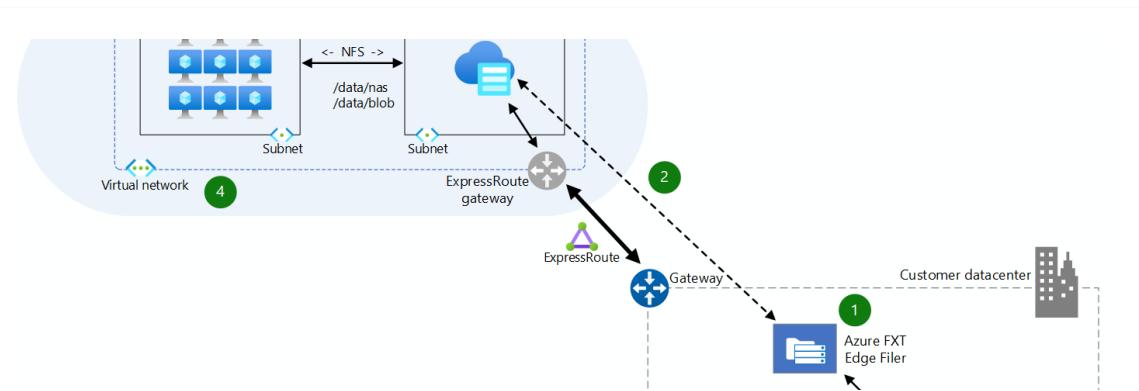
Storage architectures



HIPAA and HITRUST compliant health data AI

12/16/2019 2 min read

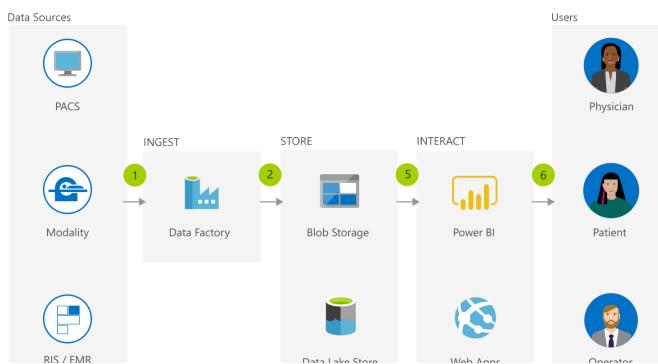
Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.



HPC Media Rendering

11/04/2020 2 min read

Optimize the media rendering process with a step-by-step HPC solution architecture from Azure that combines Azure CycleCloud and HPC Cache.



Medical Data Storage Solutions

12/16/2019 2 min read

Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security.

[view all](#)

See also

- [Microsoft Azure Storage Performance and Scalability Checklist](#)
- [Azure Storage security guide](#)
- [Best practices for using content delivery networks \(CDNs\)](#)

AWS to Azure services comparison

Article • 09/08/2023

This article helps you understand how Microsoft Azure services compare to Amazon Web Services (AWS). Whether you are planning a multicloud solution with Azure and AWS, or migrating to Azure, you can compare the IT capabilities of Azure and AWS services in all categories.

This article compares services that are roughly comparable. Not every AWS service or Azure service is listed, and not every matched service has exact feature-for-feature parity.

Azure and AWS for multicloud solutions

As the leading public cloud platforms, Azure and AWS each offer a broad and deep set of capabilities with global coverage. Yet many organizations choose to use both platforms together for greater choice and flexibility, as well as to spread their risk and dependencies with a multicloud approach. Consulting companies and software vendors might also build on and use both Azure and AWS, as these platforms represent most of the cloud market demand.

For an overview of Azure for AWS users, see [Introduction to Azure for AWS professionals](#).

Marketplace

AWS service	Azure service	Description
AWS Marketplace ↗	Azure Marketplace ↗	Easy-to-deploy and automatically configured third-party applications, including single virtual machine or multiple virtual machine solutions.

AI and machine learning

AWS service	Azure service	Description
SageMaker ↗	Machine Learning ↗	A cloud service to train, deploy, automate, and manage machine learning models.
Alexa Skills Kit ↗	Bot Framework ↗	Build and connect intelligent bots that interact with your users using text/SMS, Skype, Teams, Slack, Microsoft 365

AWS service	Azure service	Description
		mail, Twitter, and other popular services.
Lex ↗	Speech Services ↗	API capable of converting speech to text, understanding intent, and converting text back to speech for natural responsiveness.
Lex ↗	Language Understanding (LUIS) ↗	Allows your applications to understand user commands contextually.
Polly ↗, Transcribe ↗	Speech Services ↗	Enables both Speech to Text, and Text into Speech capabilities.
Rekognition ↗	Cognitive Services ↗ Computer Vision ↗	Extract information from images to categorize and process visual data. Face : Detect, identify, and analyze faces and facial expressions in photos.
Skills Kit ↗	Virtual Assistant	The Virtual Assistant Template brings together a number of best practices we've identified through the building of conversational experiences and automates integration of components that we've found to be highly beneficial to Bot Framework developers.

AI and machine learning architectures

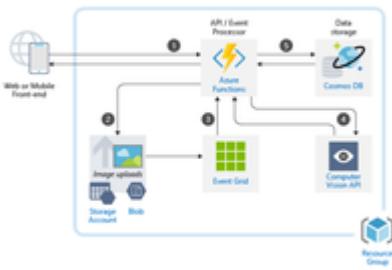
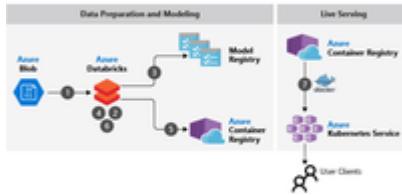


Image classification on Azure

7/05/2018 4 min read

Learn how to build image processing into your applications by using Azure services such as the Computer Vision API and Azure Functions.



Scalable personalization on Azure

5/31/2019 6 min read

[view all](#)

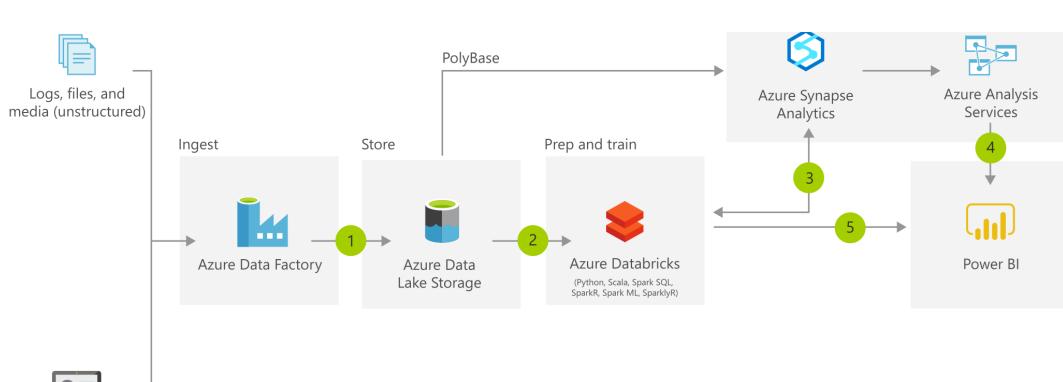
Use machine learning to automate content-based personalization for customers.

Big data and analytics

Data warehouse

AWS service	Azure service	Description
Redshift ↗	Synapse Analytics ↗	Cloud-based enterprise data warehouse (EDW) that uses massively parallel processing (MPP) to quickly run complex queries across petabytes of data.
Lake Formation ↗	Data Share ↗	A simple and safe service for sharing big data.

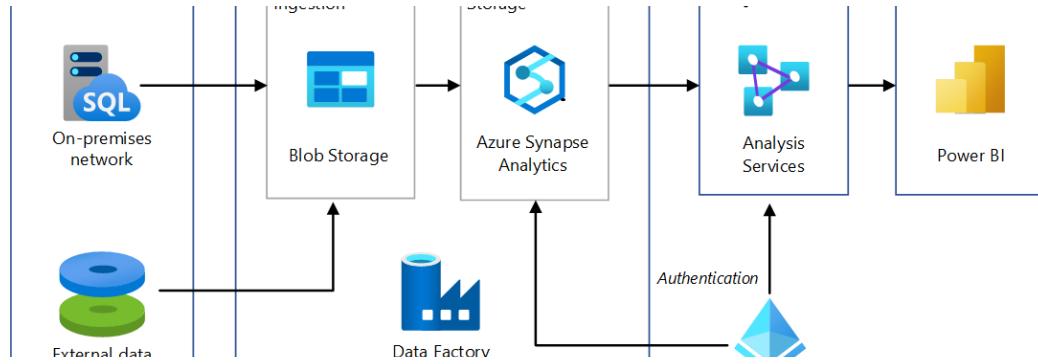
Data warehouse architectures



Modern Data Warehouse Architecture

12/16/2019 2 min read

Explore a cloud data warehouse that uses big data. Modern data warehouse brings together all your data and scales easily as your data grows.



Automated enterprise BI

6/03/2020 13 min read

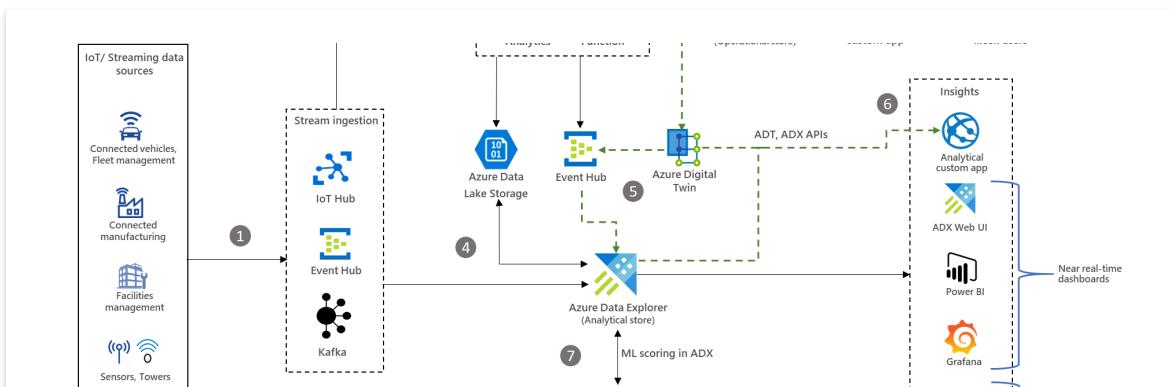
Automate an extract, load, and transform (ELT) workflow in Azure using Azure Data Factory with Azure Synapse Analytics.

[view all](#)

Time series

AWS service	Azure service	Description
Amazon Timestream ↗	Azure Data Explorer ↗	Fully managed, low latency, and distributed big data analytics platform that runs complex queries across petabytes of data. Highly optimized for log and time series data.
	Azure Time Series Insights ↗	Open and scalable end-to-end IoT analytics service. Collect, process, store, query, and visualize data at Internet of Things (IoT) scale--data that's highly contextualized and optimized for time series.

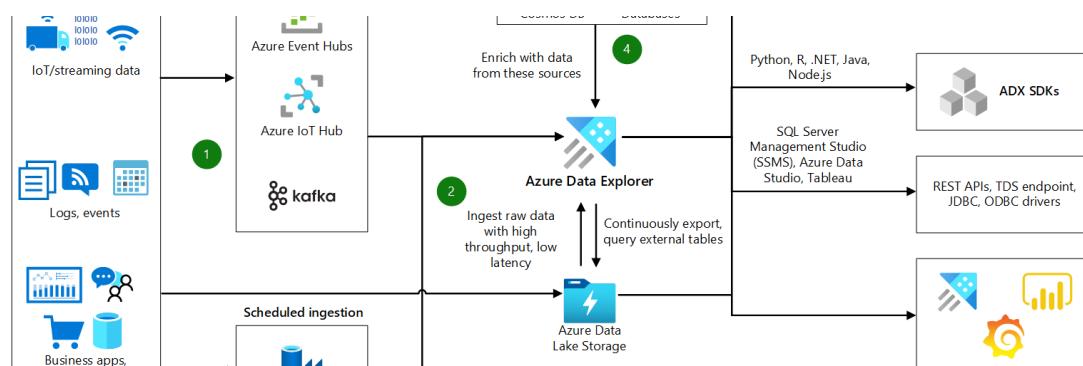
Time series architectures



IoT analytics with Azure Data Explorer

8/11/2020 3 min read

IoT Telemetry Analytics with Azure Data Explorer demonstrates near real-time analytics over fast flowing, high volume, wide variety of streaming data from IoT devices.



Azure Data Explorer interactive analytics

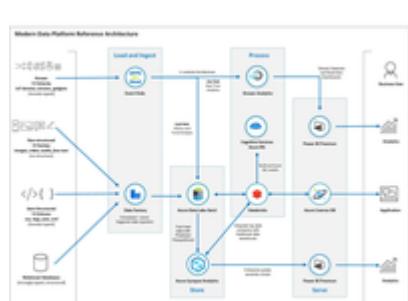
8/11/2020 3 min read

Interactive Analytics with Azure Data Explorer focuses on its integration with the rest of the data platform ecosystem.

Big data processing

AWS service	Azure service	Description
EMR ↗	Azure Data Explorer ↗	Fully managed, low latency, distributed big data analytics platform to run complex queries across petabytes of data.
EMR ↗	Databricks ↗	Apache Spark-based analytics platform.
EMR ↗	HDInsight ↗	Managed Hadoop service. Deploy and manage Hadoop clusters in Azure.
EMR ↗	Data Lake Storage ↗	Massively scalable, secure data lake functionality built on Azure Blob Storage.

Big data architectures



Azure data platform end-to-end

1/31/2020 7 min read

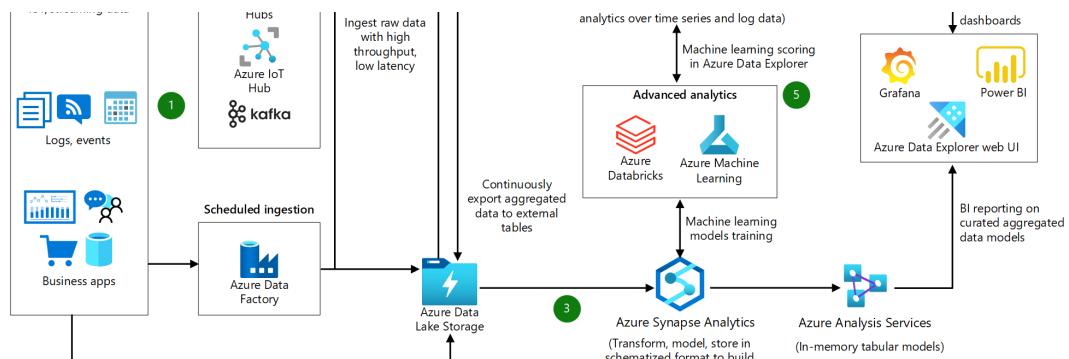
Use Azure services to ingest, process, store, serve, and visualize data from different sources.



Campaign Optimization with Azure HDInsight Spark Clusters

12/16/2019 4 min read

This solution demonstrates how to build and deploy a machine learning model with Microsoft R Server on Azure HDInsight Spark clusters to recommend actions to maximize the purchase rate of leads targeted by a campaign. This solution enables efficient handling of big data on Spark with Microsoft R Server.



Big data analytics with Azure Data Explorer

8/11/2020 3 min read

Big Data Analytics with Azure Data Explorer demonstrates Azure Data Explorer's abilities to cater to volume, velocity, and variety of data, the three V's of big data.

[view all](#)

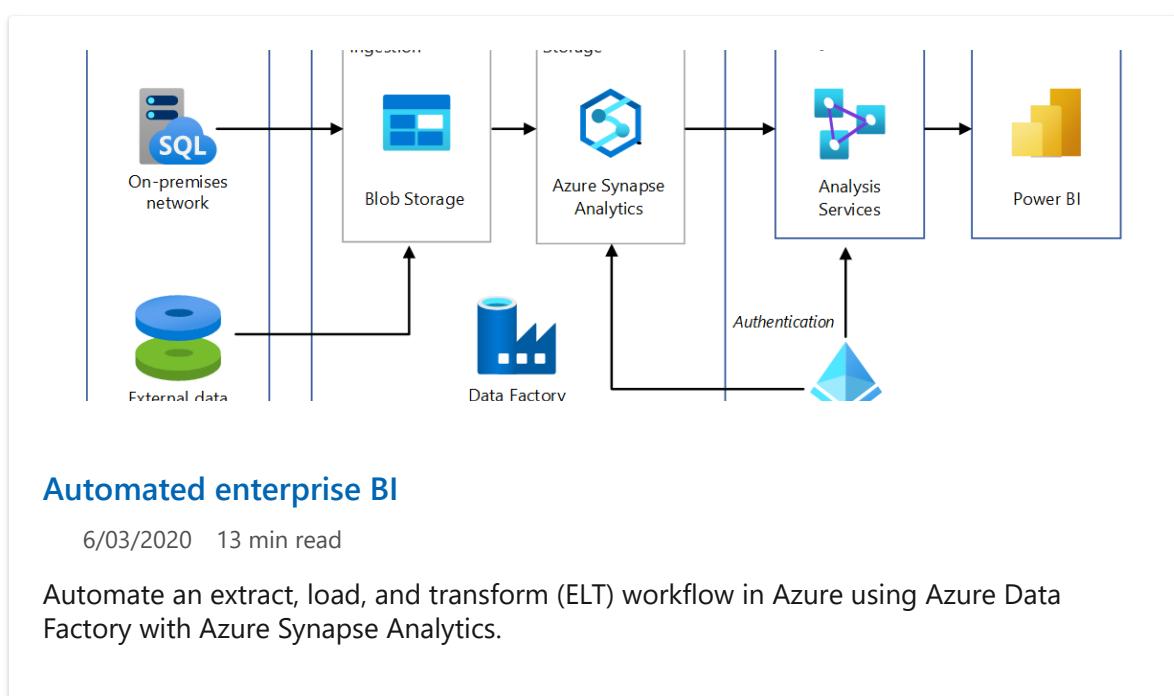
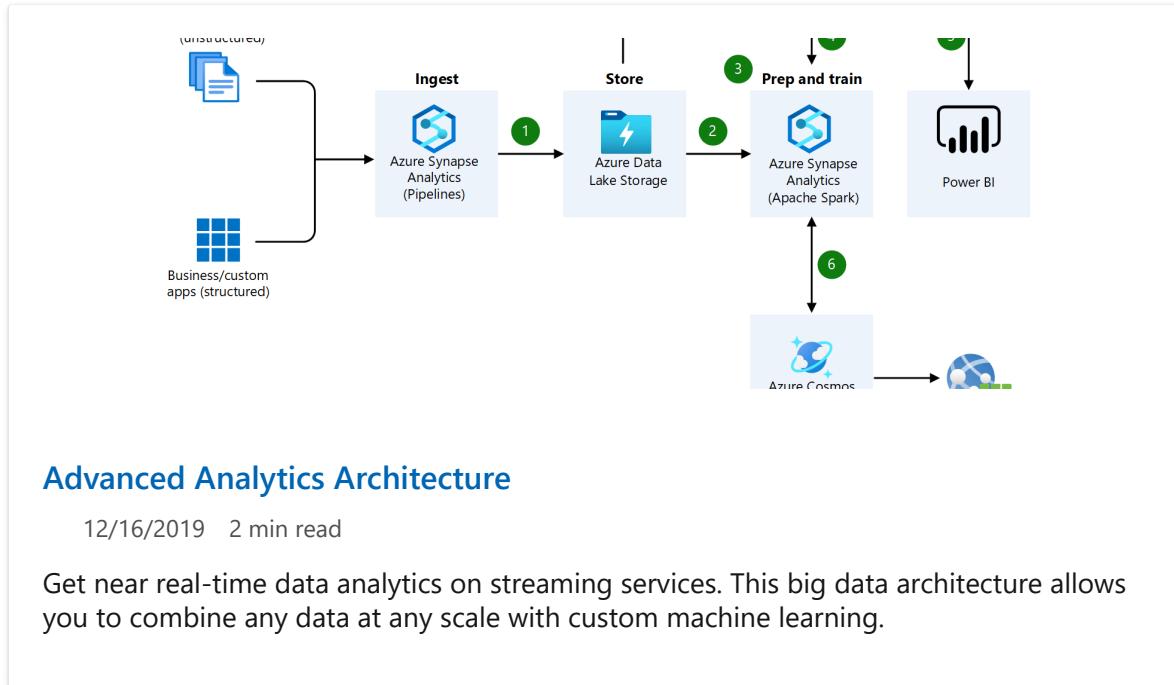
Data orchestration / ETL

AWS service	Azure service	Description
Data Pipeline ↗ , Glue ↗	Data Factory ↗	Processes and moves data between different compute and storage services, as well as on-premises data sources at specified intervals. Create, schedule, orchestrate, and manage data pipelines.
Glue ↗	Azure Purview ↗	A unified data governance service that helps you manage and govern your on-premises, multicloud, and software as a service (SaaS) data.

Analytics and visualization

AWS service	Azure service	Description
Kinesis Analytics ↗	Stream Analytics ↗	Storage and analysis platforms that create insights from large quantities of data, or data that originates from many sources.
	Azure Data Explorer ↗	
	Data Lake Analytics ↗	
	Data Lake Store ↗	
QuickSight ↗	Power BI ↗	Business intelligence tools that build visualizations, perform ad hoc analysis, and develop business insights from data.
CloudSearch ↗	Cognitive Search ↗	Delivers full-text search and related search analytics and capabilities.
Athena ↗	Data Lake Analytics ↗	Provides a serverless interactive query service that uses standard SQL for analyzing databases.
	Azure Synapse Analytics ↗	Azure Synapse Analytics is a limitless analytics service that brings together data integration, enterprise data warehousing, and big data analytics. It gives you the freedom to query data on your terms, using either serverless or dedicated resources at scale.
Elasticsearch Service ↗	Elastic on Azure ↗	Use the Elastic Stack (Elastic, Logstash, and Kibana) to search, analyze, and visualize in real time.

Analytics architectures



Create a pipeline for ingesting and analyzing text, images, sentiment, and other data from RSS news feeds using only Azure services, including Azure Cosmos DB and Azure Cognitive Services.

[view all](#)

Compute

Virtual machines and servers

Virtual machines (VMs) and servers allow users to deploy, manage, and maintain OS and other software. Users pay for what they use, with the flexibility to change sizes.

AWS service	Azure service	Description
Amazon EC2 Instance Types 	Azure Virtual Machines 	AWS and Azure on-demand VMs bill per seconds used. Although AWS instance types and Azure VM sizes have similar categories, the exact RAM, CPU, and storage capabilities differ. For information about Azure VM sizes, see Azure VM sizes .
VMware Cloud on AWS 	Azure VMware Solution 	AWS and Azure solutions let you move VMware vSphere-based workloads and environments to the cloud. Azure VMware Solution is a VMware-verified Microsoft service that runs on Azure infrastructure. You can manage existing environments with VMware solution tools, while modernizing applications with cloud native services.
AWS Parallel Cluster 	Azure CycleCloud 	Create, manage, operate, and optimize HPC and large compute clusters of any scale.

[View all the virtual machines architectures](#)

Autoscaling

Autoscaling lets you automatically change the number of VM instances. You set defined metrics and thresholds that determine when to add or remove instances.

AWS service	Azure service	Description
AWS Auto Scaling 	Virtual machine scale sets, App Service autoscale	In Azure, virtual machine scale sets let you deploy and manage identical sets of VMs. The number of sets can autoscale. App Service autoscale lets you autoscale Azure App Service applications.

[View all the autoscaling architectures](#)

Batch processing

Batch processing runs large-scale parallel and high-performance computing applications efficiently in the cloud.

AWS service	Azure service	Description
AWS Batch ↗ Azure Batch ↗ Azure Batch helps you manage compute-intensive work across a scalable collection of VMs.		

[View all the batch processing architectures](#)

Storage

Several services provide different types of data storage for VM disks.

AWS service	Azure service	Description
Disk volumes on Amazon Elastic Block Store (EBS) ↗	Data disks in Azure Blob Storage ↗.	Data disks in blob storage provide durable data storage for Azure VMs. This storage is similar to AWS EC2 instance disk volumes on EBS.
Amazon EC2 instance store ↗	Azure temporary storage	Azure temporary storage provides VMs with similar low-latency temporary read-write storage to EC2 instance storage, also called ephemeral storage.
Amazon EBS Provisioned IOPS Volume ↗	Azure premium storage	Azure supports higher performance disk I/O with premium storage. This storage is similar to AWS Provisioned IOPS storage options.
Amazon Elastic File System (EFS) ↗	Azure Files	Azure Files provides VMs with similar functionality to Amazon EFS.

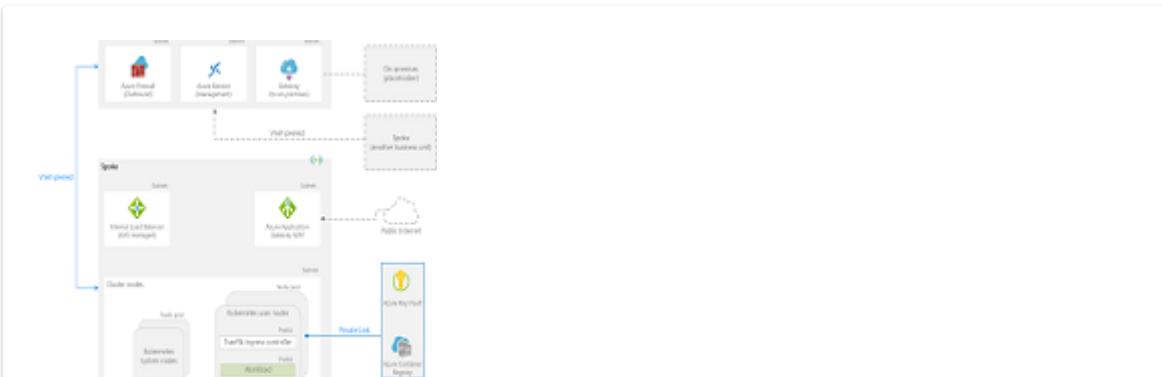
[View all the storage architectures](#)

Containers and container orchestrators

Several AWS and Azure services provide containerized application deployment and orchestration.

AWS service	Azure service	Description
Amazon Elastic Container Service (Amazon ECS) ↗ , AWS Fargate ↗	Azure Container Apps ↗	Azure Container Apps is a scalable service that lets you deploy thousands of containers without requiring access to the control plane.
Amazon Elastic Container Registry (Amazon ECR) ↗	Azure Container Registry ↗	Container registries store Docker formatted images and create all types of container deployments in the cloud.
Amazon Elastic Kubernetes Service (EKS) ↗	Azure Kubernetes Service (AKS) ↗	EKS and AKS let you orchestrate Docker containerized application deployments with Kubernetes. AKS simplifies monitoring and cluster management through auto upgrades and a built-in operations console. See Container runtime configuration for specifics on the hosting environment.
AWS App Mesh ↗	Open Service Mesh on AKS	The Open Service Mesh add-on integrates with features provided by Azure as well as open source projects.

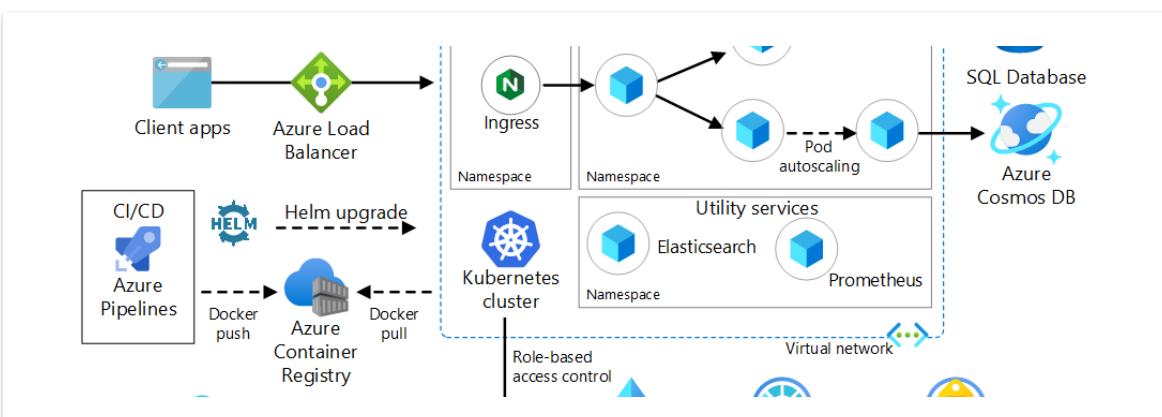
Example container architectures



Baseline architecture on Azure Kubernetes Service (AKS)

07/20/2020 37 min read

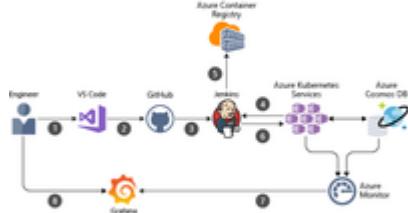
Deploy a baseline infrastructure that deploys an AKS cluster with focus on security.



Microservices architecture on Azure Kubernetes Service (AKS)

5/07/2020 17 min read

Deploy a microservices architecture on Azure Kubernetes Service (AKS)



CI/CD pipeline for container-based workloads

7/05/2018 7 min read

Build a DevOps pipeline for a Node.js web app with Jenkins, Azure Container Registry, Azure Kubernetes Service, Azure Cosmos DB, and Grafana.

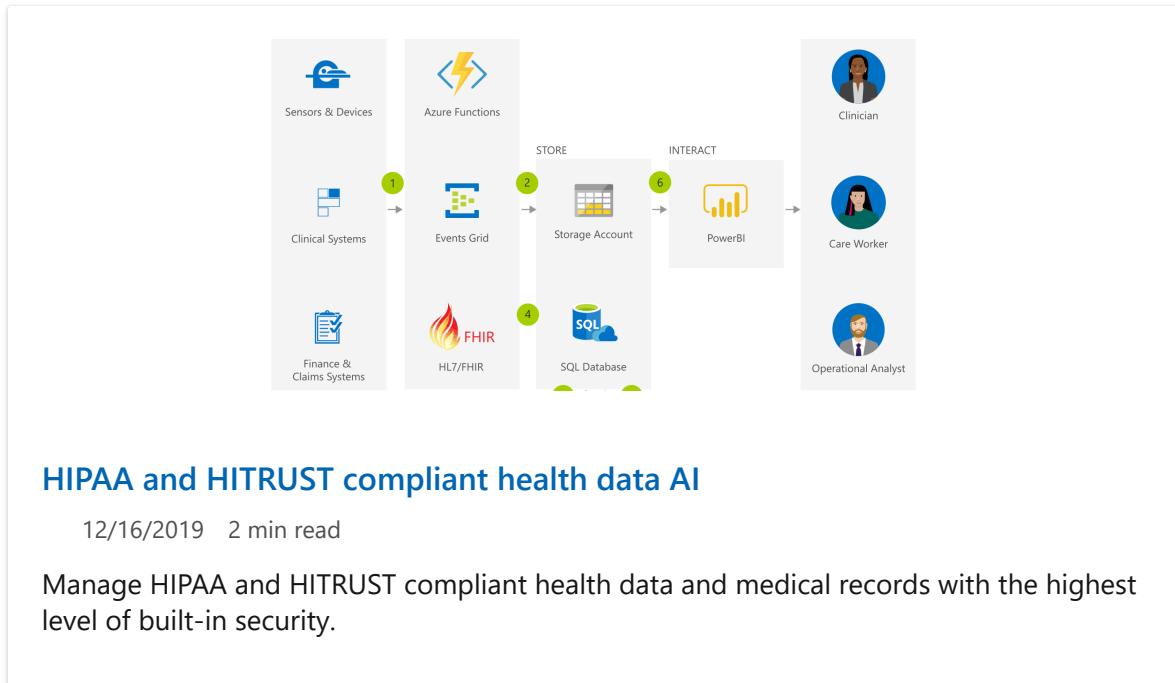
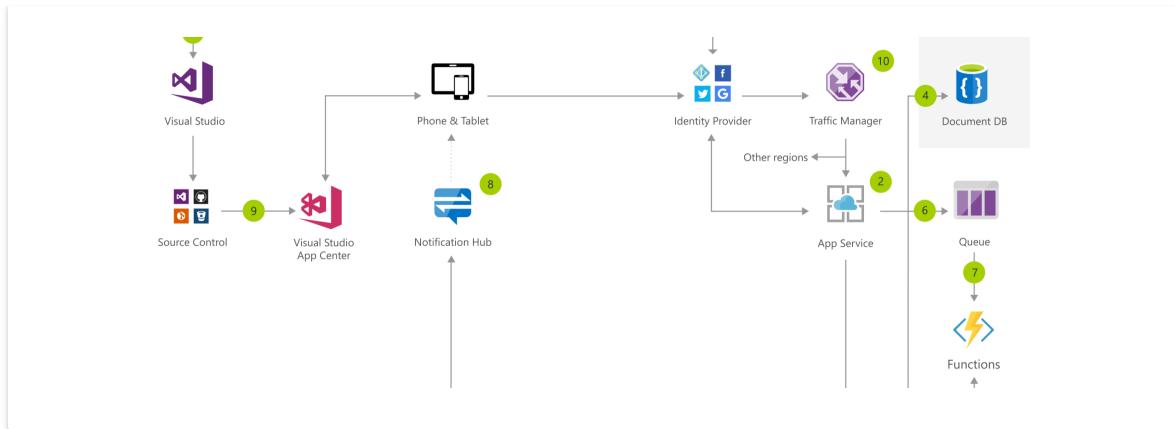
[View all the container architectures](#)

Serverless computing

Serverless computing lets you integrate systems and run backend processes without provisioning or managing servers.

AWS service	Azure service	Description
AWS Lambda	Azure Functions , WebJobs in Azure App Service	Azure Functions is the primary equivalent of AWS Lambda in providing serverless, on-demand code. AWS Lambda functionality also overlaps with Azure WebJobs, which let you schedule or continuously run background tasks.

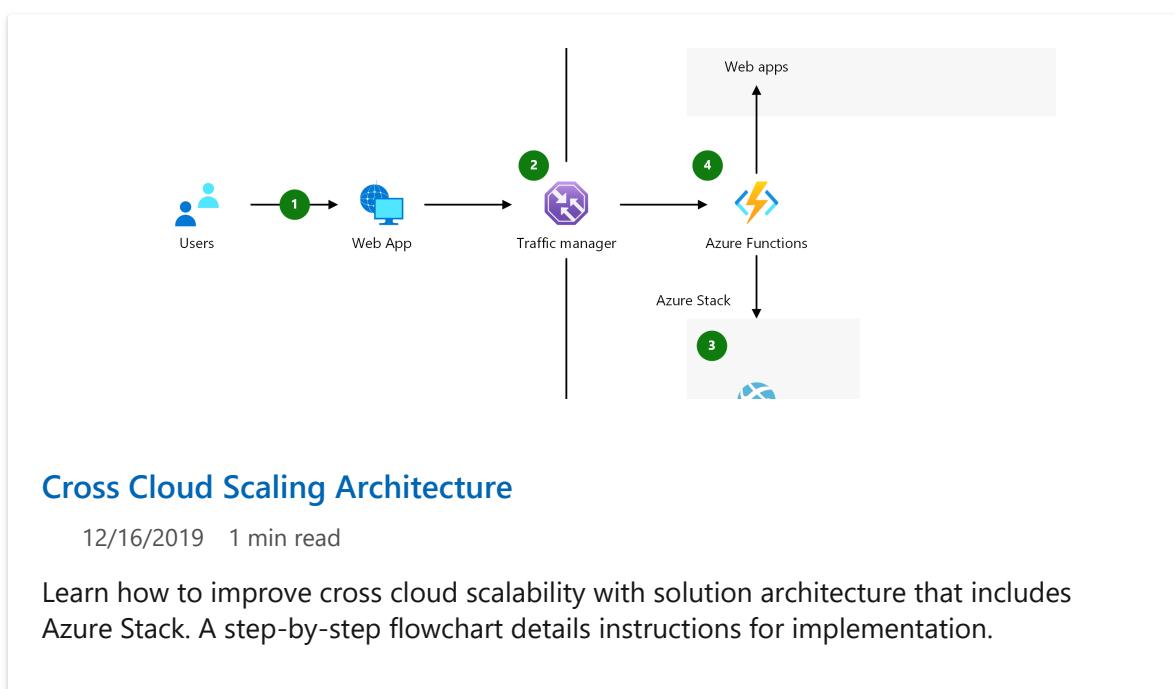
Example serverless architectures



HIPAA and HITRUST compliant health data AI

12/16/2019 2 min read

Manage HIPAA and HITRUST compliant health data and medical records with the highest level of built-in security.



Cross Cloud Scaling Architecture

12/16/2019 1 min read

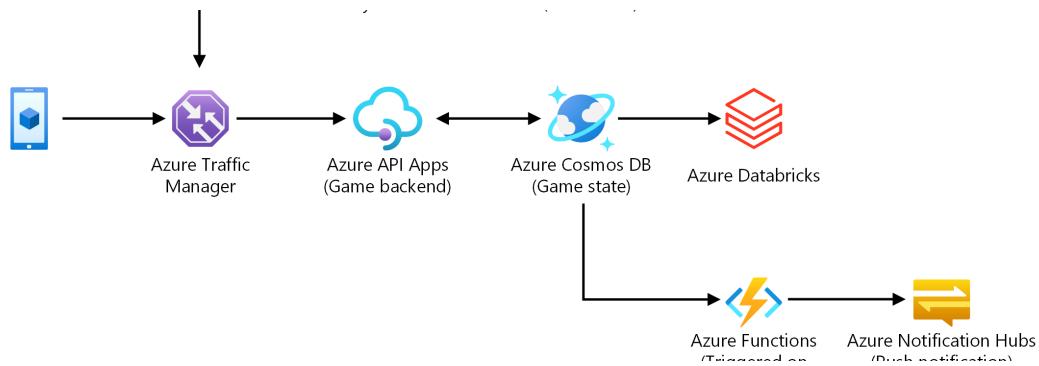
Learn how to improve cross cloud scalability with solution architecture that includes Azure Stack. A step-by-step flowchart details instructions for implementation.

[View all the serverless architectures](#)

Database

Type	AWS Service	Azure Service	Description
Relational database	RDS ↗	SQL Database ↗	Managed relational database services in which resiliency, scale and maintenance are primarily handled by the Azure platform.
		Database for MySQL ↗	
		Database for PostgreSQL ↗	
Serverless relational database	Amazon Aurora Serverless ↗	Database serverless	Database offerings that automatically scales compute based on the workload demand. You're billed per second for the actual compute used (Azure SQL)/data that's processed by your queries (Azure Synapse Analytics Serverless).
		Serverless SQL pool in Azure Synapse Analytics	
NoSQL	DynamoDB ↗ (Key-Value)	Azure Cosmos DB ↗	Azure Cosmos DB is a globally distributed, multi-model database that natively supports multiple data models including key-value pairs, documents, graphs, and columnar.
	SimpleDB ↗ Amazon DocumentDB ↗ (Document)		
	Amazon Neptune ↗ (Graph)		
Caching	ElastiCache ↗ Amazon MemoryDB for Redis ↗	Cache for Redis ↗	An in-memory-based, distributed caching service that provides a high-performance store that's typically used to offload nontransactional work from a database.
Database migration	Database Migration Service ↗	Database Migration Service ↗	A service that executes the migration of database schema and data from one database format to a specific database technology in the cloud.

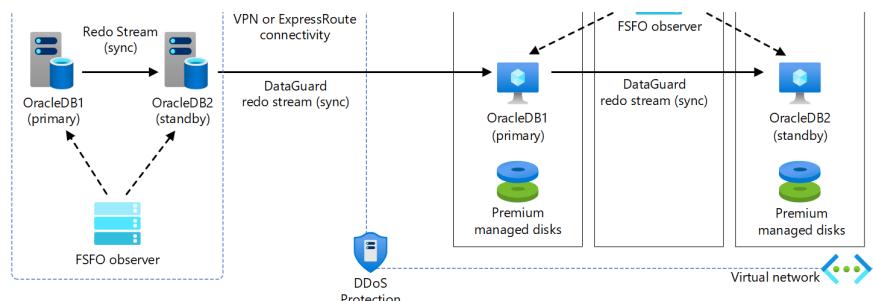
Database architectures



Gaming using Azure Cosmos DB

12/16/2019 1 min read

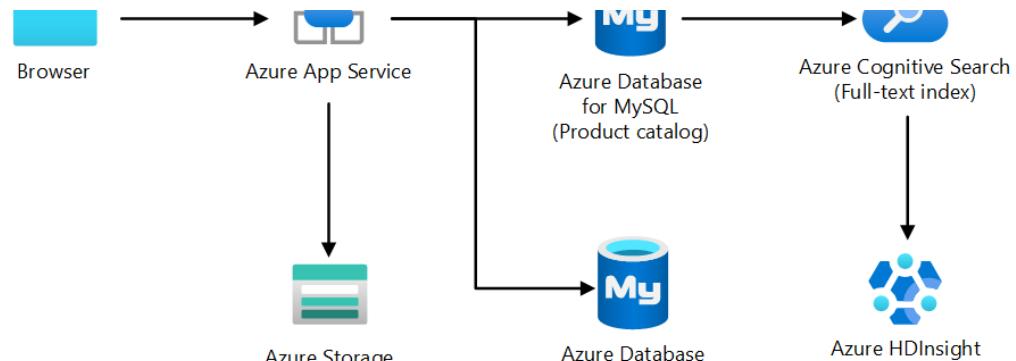
Elastically scale your database to accommodate unpredictable bursts of traffic and deliver low-latency multi-player experiences on a global scale.



Oracle Database Migration to Azure

12/16/2019 2 min read

Oracle DB migrations can be accomplished in multiple ways. This architecture covers one of these options wherein Oracle Active Data Guard is used to migrate the Database.



Retail and e-commerce using Azure MySQL

12/16/2019 1 min read

Build secure and scalable e-commerce solutions that meet the demands of both customers and business using Azure Database for MySQL.

[view all](#)

DevOps and application monitoring

AWS service	Azure service	Description
CloudWatch <small>↗</small> , X-Ray <small>↗</small>	Monitor <small>↗</small>	Comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.
CodeDeploy <small>↗</small>	DevOps <small>↗</small>	A cloud service for collaborating on code development.
CodeCommit <small>↗</small>		
CodePipeline <small>↗</small>		
Developer Tools <small>↗</small>	Developer Tools <small>↗</small>	Collection of tools for building, debugging, deploying, diagnosing, and managing multiplatform scalable apps and services.
CodeBuild <small>↗</small>	DevOps Pipeline <small>↗</small>	Fully managed build service that supports continuous integration and deployment.
	Github Actions <small>↗</small>	
Command Line Interface <small>↗</small>	CLI	Built on top of the native REST API across all cloud services, various programming language-specific wrappers provide easier ways to create solutions.
	PowerShell	
eksctl <small>↗</small>	az aks	Manage Azure Kubernetes Service using these Azure CLI commands.
AWS CloudShell <small>↗</small>	Azure Cloud Shell	Azure Cloud Shell is an interactive, authenticated, browser-accessible shell for managing Azure resources. It gives you the flexibility to choose the shell experience that best suits the way you work, either Bash or PowerShell.
OpsWorks (Chef-based) <small>↗</small>	Automation <small>↗</small>	Configures and operates applications of all shapes and sizes, and provides templates to create and manage a collection of resources.
CloudFormation <small>↗</small>	Resource Manager <small>↗</small>	Provides a way for users to automate the manual, long-running, error-prone, and frequently repeated IT tasks.
	Bicep	
	VM extensions	

AWS service	Azure service	Description
	Azure Automation ↗	

DevOps architectures

The diagram illustrates a CI/CD pipeline. It starts with GitHub (labeled 2) pushing code to Jenkins on an Azure VM (labeled 3). Jenkins then triggers Jenkins build agents in AKS (labeled 4). The build agents interact with Azure Cosmos DB (labeled 7). Finally, the results are monitored by Azure Monitor (labeled 7).

Container CI/CD using Jenkins and Kubernetes on Azure Kubernetes Service (AKS)

12/16/2019 2 min read

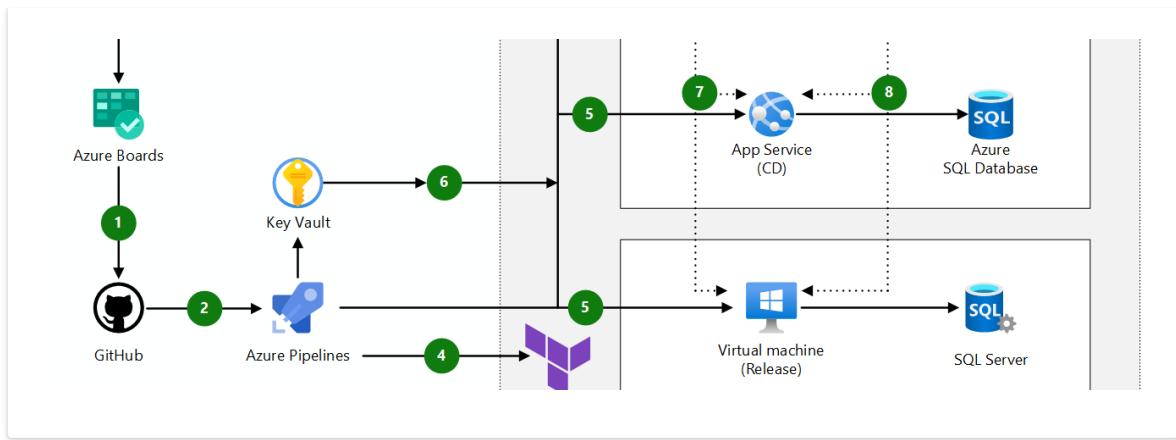
Containers make it easy for you to continuously build and deploy applications. By orchestrating the deployment of those containers using Azure Kubernetes Service (AKS), you can achieve replicable, manageable clusters of containers.

This diagram shows a Jenkins server running on an Azure VM. The Jenkins server interacts with a Virtual network (Subnet) and a public IP address. It also connects to Jenkins build agents (multiple VMs) via a private endpoint. Various Azure services are integrated: authentication through Azure Active Directory; storage via Managed Disks and Azure Blob Storage; monitoring via Azure Monitor; and security via Azure Key Vault. Jenkins also manages its own Data and OS storage.

Run a Jenkins server on Azure

11/19/2020 6 min read

Recommended architecture that shows how to deploy and operate a scalable, enterprise-grade Jenkins server on Azure secured with single sign-on (SSO).



[view all DevOps in a hybrid environment](#)

12/16/2019 3 min read

The tools provided in Azure allow for the implementation of a DevOps strategy that can manage both cloud and on-premises environments in tandem.

AWS service	Azure service	Description
IoT Core ↗	IoT Hub ↗	A cloud gateway for managing bidirectional communication with billions of IoT devices, securely and at scale.
Greengrass ↗	IoT Edge ↗	Deploy cloud intelligence directly onto IoT devices, catering to on-premises scenarios.
Kinesis Firehose ↗ , Kinesis Streams ↗	Event Hubs ↗	Services that facilitate the mass ingestion of events (messages), typically from devices and sensors. The data can then be processed in real-time micro-batches or be written to storage for further analysis.
IoT Things Graph ↗	Digital Twins ↗	Services you can use to create digital representations of real-world things, places, business processes, and people. Use these services to gain insights, drive the creation of better products and new customer experiences, and optimize operations and costs.

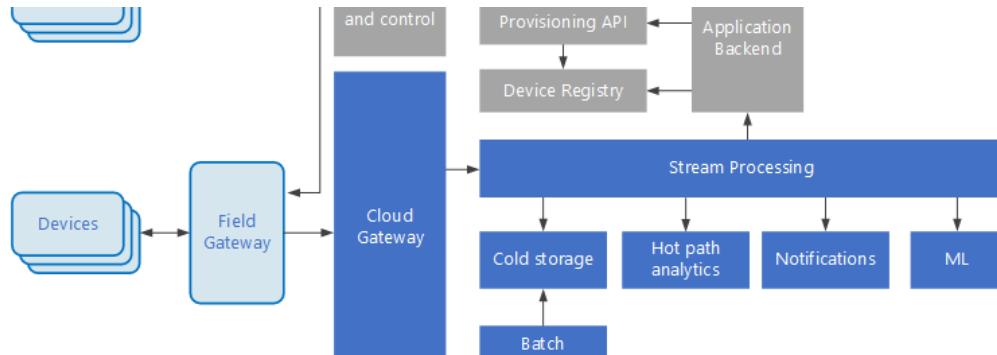
IoT architectures



IoT Architecture ◆ Azure IoT Subsystems

12/16/2019 1 min read

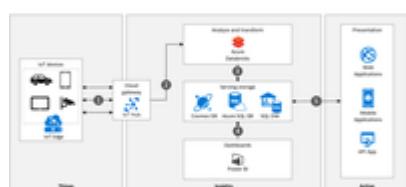
Learn about our recommended IoT application architecture that supports hybrid cloud and edge computing. A flowchart details how the subsystems function within the IoT application.



Azure IoT reference architecture

9/10/2020 12 min read

Recommended architecture for IoT applications on Azure using PaaS (platform-as-a-service) components



Process real-time vehicle data using IoT

11/17/2020 5 min read

This example builds a real-time data ingestion/processing pipeline to ingest and process messages from IoT devices into a big data analytic platform in Azure.

[view all](#)

Management and governance

AWS service	Azure service	Description
AWS Organizations	Management Groups	Azure management groups help you organize your resources and subscriptions.
AWS Well-Architected Tool	Azure Well-Architected Review	Examine your workload through the lenses of reliability, cost management, operational excellence, security, and performance efficiency.
Trusted Advisor	Advisor	Provides analysis of cloud resource configuration and security, so that subscribers can ensure they're making use of best practices and optimum configurations.
AWS Billing and Cost Management	Azure Cost Management and Billing	Azure Cost Management and Billing helps you understand your Azure invoice (bill), manage your billing account and subscriptions, monitor and control Azure spending, and optimize resource use.
Cost and Usage Reports	Usage Details API	Services to help generate, monitor, forecast, and share billing data for resource usage by time, organization, or product resources.
Management Console	Portal	A unified management console that simplifies building, deploying, and operating your cloud resources.
Application Discovery Service	Migrate	Assesses on-premises workloads for migration to Azure, performs performance-based sizing, and provides cost estimations.
Systems Manager	Monitor	Comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.
Personal Health Dashboard	Resource Health	Provides detailed information about the health of resources, as well as recommended actions for maintaining resource health.
CloudTrail	Activity log	The Activity log is a platform log in Azure that provides insight into subscription-level events, such as when a resource is modified or when a virtual machine is started.
CloudWatch	Application Insights	A feature of Azure Monitor, Application Insights is an extensible Application Performance Management (APM) service for developers and DevOps professionals, which provides telemetry insights and information, in order to better understand how applications are performing and to identify areas for optimization.
Config	Application Change Analysis	Application Change Analysis detects various types of changes, from the infrastructure layer all the way to

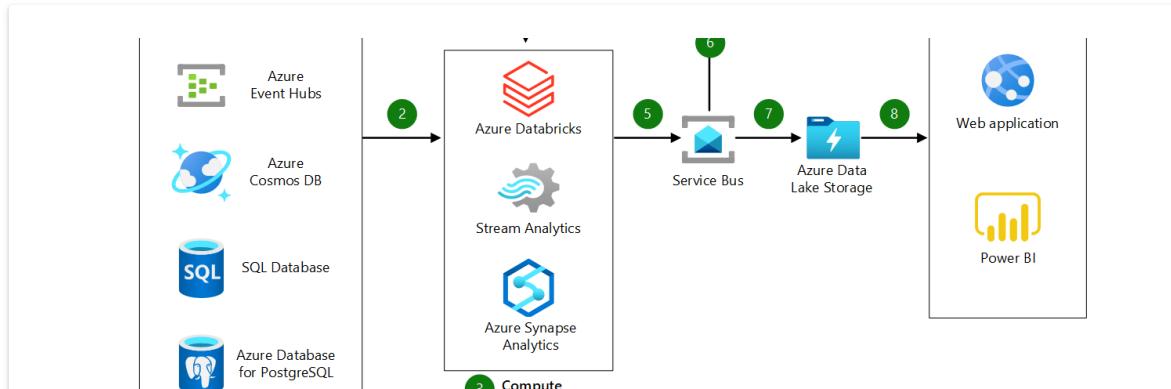
AWS service	Azure service	Description
		application deployment.
Cost Explorer ↗	Cost Management ↗	Optimize costs while maximizing cloud potential.
Control Tower ↗	Azure Lighthouse	Set up and govern a multi account/subscription environment.
Resource Groups and Tag Editor ↗	Resource Groups and Tags	A Resource Group is a container that holds related resources for an Azure solution. Apply tags to your Azure resources to logically organize them by categories.
AWS AppConfig ↗	Azure App Configuration	Azure App Configuration is a managed service that helps developers centralize their application and feature settings simply and securely.
Service Catalog ↗	Azure Managed Applications	Offers cloud solutions that are easy for consumers to deploy and operate.
SDKs and tools ↗	SDKs and tools ↗	Manage and interact with Azure services the way you prefer, programmatically from your language of choice, by using the Azure SDKs, our collection of tools, or both.

Messaging and eventing

AWS service	Azure service	Description
Simple Queue Service (SQS) ↗	Queue Storage ↗	Provides a managed message queueing service for communicating between decoupled application components.
Simple Notification Service (SNS) ↗	Service Bus ↗	Supports a set of cloud-based, message-oriented middleware technologies, including reliable message queuing and durable publish/subscribe messaging.
Amazon EventBridge ↗	Event Grid ↗	A fully managed event routing service that allows for uniform event consumption using a publish/subscribe model.
Amazon Kinesis ↗	Event Hubs ↗	A fully managed, real-time data ingestion service. Stream millions of events per second, from any source, to build dynamic data pipelines and to immediately respond to business challenges.
Amazon MQ ↗	Service Bus	Service Bus Premium is fully compliant with the Java/Jakarta EE Java Message Service (JMS) 2.0 API. Service Bus Standard

AWS service	Azure service	Description
		supports the JMS 1.1 subset focused on queues.

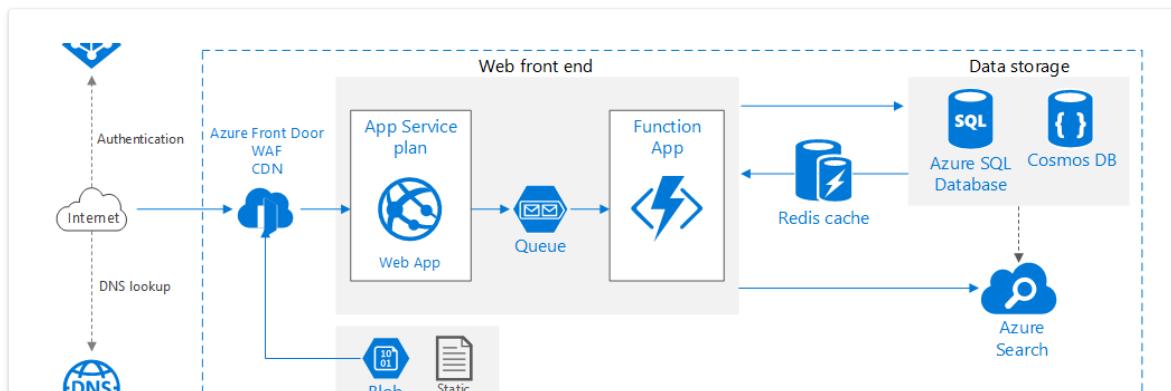
Messaging architectures



Anomaly Detector Process

12/16/2019 1 min read

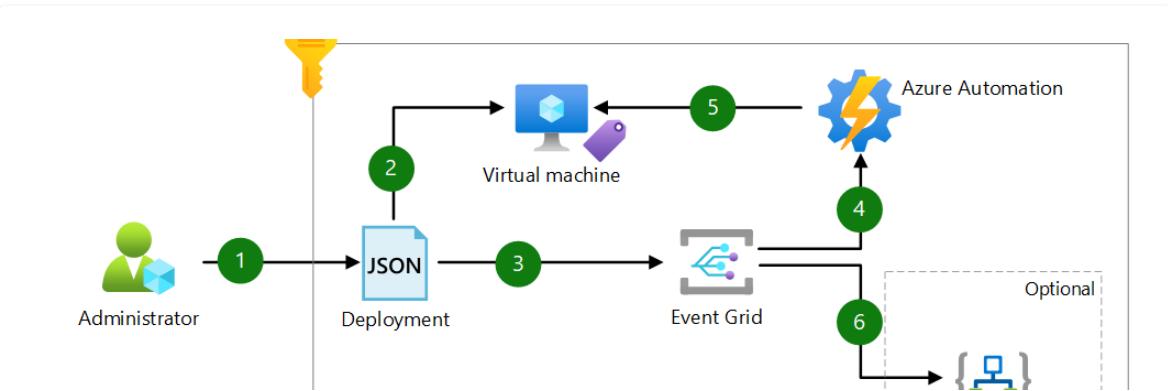
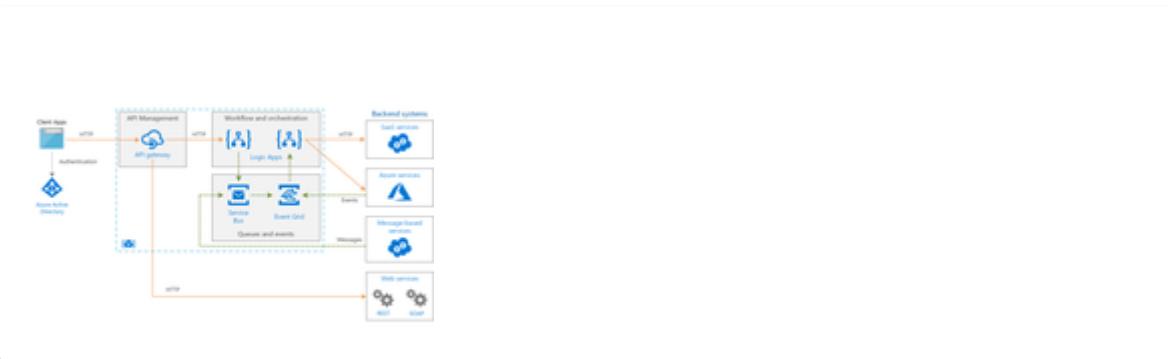
Learn more about Anomaly Detector with a step-by-step flowchart that details the process. See how anomaly detection models are selected with time-series data.



Scalable web application

10/03/2019 7 min read

Use the proven practices in this reference architecture to improve scalability and performance in an Azure App Service web application..



Ops automation using Event Grid

12/16/2019 1 min read

Event Grid allows you to speed automation and simplify policy enforcement. For example, Event Grid can notify Azure Automation when a virtual machine is created, or a SQL Database is spun up. These events can be used to automatically check that service configurations are compliant, put metadata into operations tools, tag virtual machines, ...

Mobile services

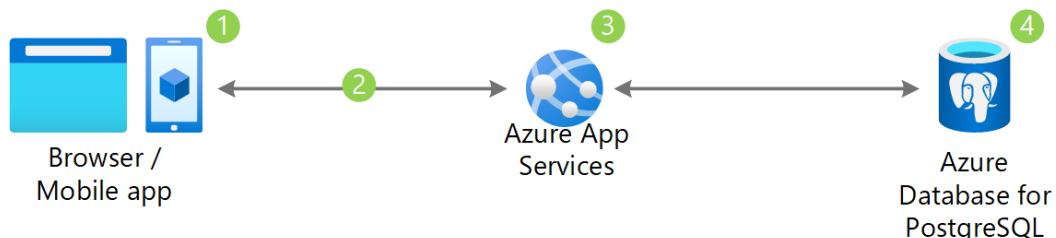
AWS service	Azure service	Description
Mobile Hub ↗	App Center ↗ Xamarin Apps ↗	Provides backend mobile services for rapid development of mobile solutions, identity management, data synchronization, and storage and notifications across devices.
Mobile SDK ↗	App Center ↗	Provides the technology to rapidly build cross-platform and native apps for mobile devices.
Device Farm ↗	App Center ↗	Provides services to support testing mobile applications.
Mobile Analytics ↗	App Center ↗	Supports monitoring, and feedback collection for the debugging and analysis of a mobile application service quality.

Device Farm

The AWS Device Farm provides cross-device testing services. In Azure, [Visual Studio App Center](#) provides similar cross-device front-end testing for mobile devices.

In addition to front-end testing, the [Azure DevTest Labs](#) provides back-end testing resources for Linux and Windows environments.

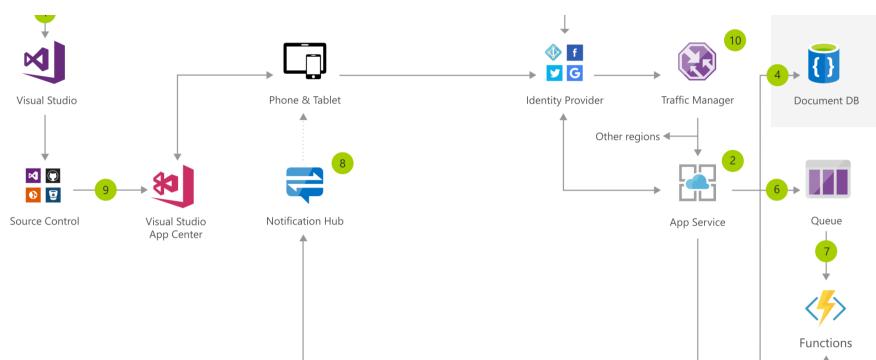
Mobile architectures



Scalable web and mobile applications using Azure Database for PostgreSQL

12/16/2019 1 min read

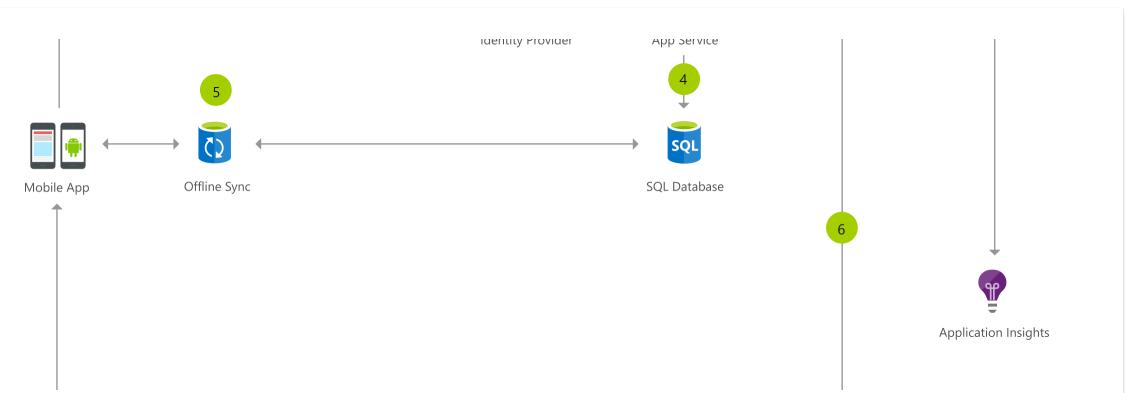
Use Azure Database for PostgreSQL to rapidly build engaging, performant, and scalable cross-platform and native apps for iOS, Android, Windows, or Mac.



Social App for Mobile and Web with Authentication

12/16/2019 3 min read

View a detailed, step-by-step diagram depicting the build process and implementation of the mobile client app architecture that offers social image sharing with a companion web app and authentication abilities, even while offline.



Task-Based Consumer Mobile App

12/16/2019 3 min read

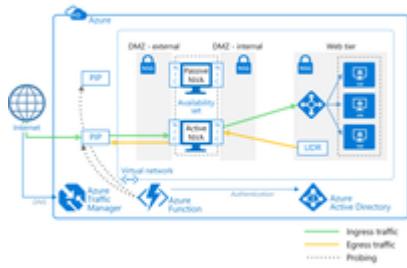
Learn how the task-based consumer mobile app architecture is created with a step-by-step flow chart that shows the integration with Azure App Service Mobile Apps, Visual Studio, and Xamarin to simplify the build process.

Networking

Area	AWS service	Azure service	Description
Cloud virtual networking	Virtual Private Cloud (VPC)	Virtual Network	Provides an isolated, private environment in the cloud. Users have control over their virtual networking environment, including selection of their own IP address range, creation of subnets, and configuration of route tables and network gateways.
NAT gateways	NAT Gateways	Virtual Network NAT	A service that simplifies outbound-only Internet connectivity for virtual networks. When configured on a subnet, all outbound connectivity uses your specified static public IP addresses. Outbound connectivity is possible without a load balancer or public IP addresses directly attached to virtual machines.
Cross-premises connectivity	VPN Gateway	VPN Gateway	Connects Azure virtual networks to other Azure virtual networks, or customer on-premises networks (Site To Site). Allows end users to connect to Azure services through VPN tunneling (Point To Site).
DNS management	Route 53	DNS	Manage your DNS records using the same credentials and billing and support contract as your other Azure services
DNS-based routing	Route 53	Traffic Manager	A service that hosts domain names, plus routes users to Internet applications,

Area	AWS service	Azure service	Description
			connects user requests to datacenters, manages traffic to apps, and improves app availability with automatic failover.
Dedicated network	Direct Connect ↗ ExpressRoute ↗		Establishes a dedicated, private network connection from a location to the cloud provider (not over the Internet).
Load balancing	Network Load Balancer ↗	Load Balancer ↗	Azure Load Balancer load balances traffic at layer 4 (TCP or UDP). Standard Load Balancer also supports cross-region or global load balancing.
Application-level load balancing	Application Load Balancer ↗	Application Gateway ↗	Application Gateway is a layer 7 load balancer. It supports SSL termination, cookie-based session affinity, and round robin for load-balancing traffic.
Route table	Custom Route Tables ↗	User Defined Routes	Custom, or user-defined (static) routes to override default system routes, or to add more routes to a subnet's route table.
Private link	PrivateLink ↗	Azure Private Link ↗	Azure Private Link provides private access to services that are hosted on the Azure platform. This keeps your data on the Microsoft network.
Private PaaS connectivity	VPC endpoints ↗ Private Endpoint		Private Endpoint provides secured, private connectivity to various Azure platform as a service (PaaS) resources, over a backbone Microsoft private network.
Virtual network peering	VPC Peering ↗	VNET Peering ↗	VNet peering is a mechanism that connects two virtual networks (VNets) in the same region through the Azure backbone network. Once peered, the two virtual networks appear as one for all connectivity purposes.
Content delivery networks	CloudFront ↗	Front Door ↗	Azure Front Door is a modern cloud content delivery network (CDN) service that delivers high performance, scalability, and secure user experiences for your content and applications.
Network Monitoring	VPC Flow Logs ↗	Azure Network Watcher	Azure Network Watcher allows you to monitor, diagnose, and analyze the traffic in Azure Virtual Network.

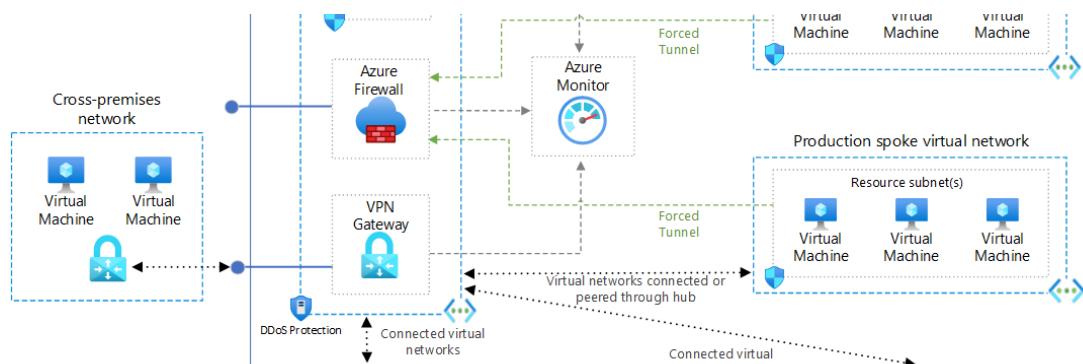
Networking architectures



Deploy highly available NVAs

12/08/2018 7 min read

Learn how to deploy network virtual appliances for high availability in Azure. This article includes example architectures for ingress, egress, and both.



Hub-spoke network topology in Azure

9/30/2020 7 min read

Learn how to implement a hub-spoke topology in Azure, where the hub is a virtual network and the spokes are virtual networks that peer with the hub.



Implement a secure hybrid network

1/07/2020 9 min read

See a secure hybrid network that extends an on-premises network to Azure with a perimeter network between the on-premises network and an Azure virtual network.

[view all](#)

Security, identity, and access

Authentication and authorization

AWS service	Azure service	Description
Identity and Access Management (IAM) ↗	Azure Active Directory ↗	Allows users to securely control access to services and resources while offering data security and protection. Create and manage users and groups, and use permissions to allow and deny access to resources.
Identity and Access Management (IAM) ↗	Azure role-based access control	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.
Organizations ↗	Subscription Management + Azure RBAC	Security policy and role management for working with multiple accounts.
Multi-Factor Authentication ↗	Azure Active Directory ↗	Safeguard access to data and applications, while meeting user demand for a simple sign-in process.
Directory Service ↗	Azure Active Directory Domain Services ↗	Provides managed domain services, such as domain join, group policy, LDAP, and Kerberos/NTLM authentication, which are fully compatible with Windows Server Active Directory.
Cognito ↗	Azure Active Directory External Identities ↗	A highly available, global, identity management service for consumer-facing applications that scales to hundreds of millions of identities.
AWS Config ↗	Policy ↗	Azure Policy is a service in Azure that you use to create, assign, and manage policies. These policies enforce different rules and effects over your resources, so those resources stay compliant with your corporate standards and service level agreements.
Organizations ↗	Management Groups	Azure management groups provide a level of scope above subscriptions. You organize subscriptions into containers called "management groups" and apply your governance conditions to the management groups. All

AWS service	Azure service	Description
		subscriptions within a management group automatically inherit the conditions applied to the management group. Management groups give you enterprise-grade management at a large scale, no matter what type of subscriptions you have.

Encryption

AWS service	Azure service	Description
Server-side encryption with Amazon S3 Key Management Service	Azure Storage Service Encryption	Helps you protect and safeguard your data and meet your organizational security and compliance commitments.
Key Management Service (KMS) , CloudHSM	Key Vault	Provides security solution and works with other services by providing a way to manage, create, and control encryption keys stored in hardware security modules (HSM).

Firewall

AWS service	Azure service	Description
Web Application Firewall	Web Application Firewall	A firewall that protects web applications from common web exploits.
AWS Network Firewall	Firewall	Provides inbound protection for non-HTTP/S protocols, outbound network-level protection for all ports and protocols, and application-level protection for outbound HTTP/S.

Security

AWS service	Azure service	Description
Inspector	Defender for Cloud	An automated security assessment service that improves the security and compliance of applications. Automatically assess applications for vulnerabilities or deviations from best practices.
Certificate Manager	App Service Certificates available on the Portal	Service that allows customers to create, manage, and consume certificates seamlessly in the cloud.

AWS service	Azure service	Description
GuardDuty ↗	Microsoft Sentinel ↗	Detect and investigate advanced attacks on-premises and in the cloud.
Artifact ↗	Service Trust Portal ↗	Provides access to audit reports, compliance guides, and trust documents from across cloud services.
Shield ↗	DDoS Protection Service	Provides cloud services with protection from distributed denial of services (DDoS) attacks.

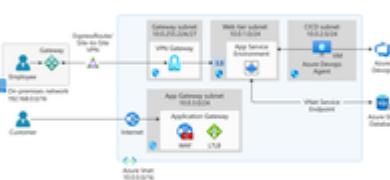
Security architectures



Real-time fraud detection

7/05/2018 4 min read

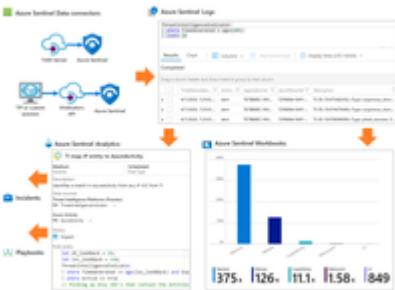
Detect fraudulent activity in real-time using Azure Event Hubs and Stream Analytics.



Securely managed web applications

5/09/2019 8 min read

Learn about deploying secure applications using the Azure App Service Environment, the Azure Application Gateway service, and Web Application Firewall.



Threat indicators for cyber threat intelligence in Azure Sentinel

4/13/2020 13 min read

Import threat indicators, view logs, create rules to generate security alerts and incidents, and visualize threat intelligence data with Azure Sentinel.
[view all](#)

Storage

Object storage

AWS service	Azure service	Description
Simple Storage Services (S3) ↗	Blob storage	Object storage service, for use cases including cloud applications, content distribution, backup, archiving, disaster recovery, and big data analytics.

Virtual server disks

AWS service	Azure service	Description
Elastic Block Store (EBS) ↗	managed disks ↗	SSD storage optimized for I/O intensive read/write operations. For use as high-performance Azure virtual machine storage.

Shared files

AWS service	Azure service	Description
Elastic File System ↗	Files ↗	Provides a simple interface to create and configure file systems quickly, and share common files. Can be used with traditional protocols that access files over a network.

Archiving and backup

AWS service	Azure service	Description
S3 Infrequent Access (IA) ↗	Storage cool tier	Cool storage is a lower-cost tier for storing data that is infrequently accessed and long-lived.
S3 Glacier ↗, Deep Archive	Storage archive access tier	Archive storage has the lowest storage cost and higher data retrieval costs compared to hot and cool storage.
Backup ↗	Backup ↗	Back up and recover files and folders from the cloud, and provide offsite protection against data loss.

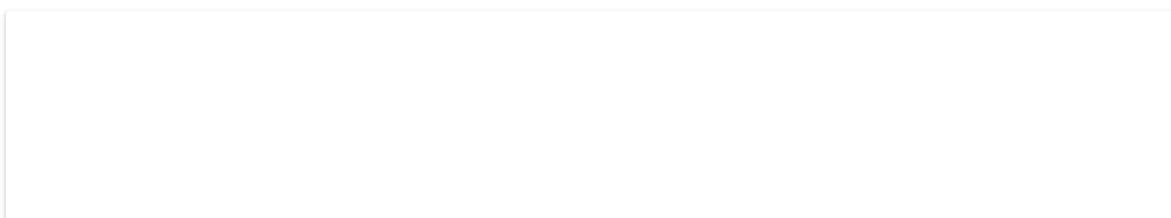
Hybrid storage

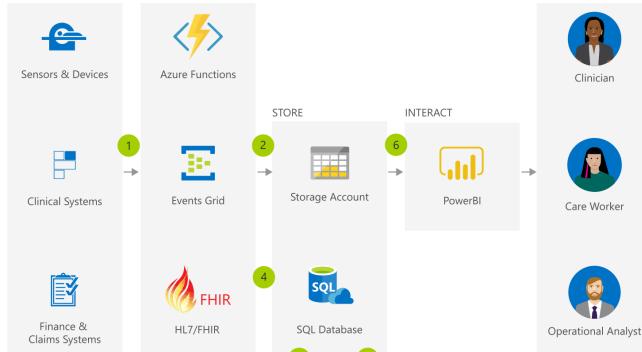
AWS service	Azure service	Description
Storage Gateway ↗	StorSimple ↗	Integrates on-premises IT environments with cloud storage. Automates data management and storage, plus supports disaster recovery.
DataSync ↗	File Sync	Azure Files can be deployed in two main ways: by directly mounting the serverless Azure file shares or by caching Azure file shares on-premises using Azure File Sync.

Bulk data transfer

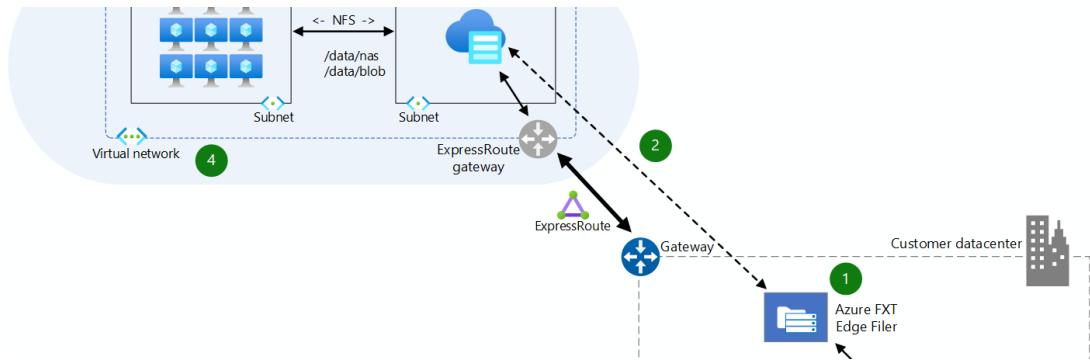
AWS service	Azure service	Description
Import/Export Disk ↗	Import/Export	A data transport solution that uses secure disks and appliances to transfer large amounts of data. Also offers data protection during transit.
Import/Export Snowball ↗, Snowball Edge ↗, Snowmobile ↗	Data Box ↗	Petabyte- to exabyte-scale data transport solution that uses secure data storage devices to transfer large amounts of data to and from Azure.

Storage architectures





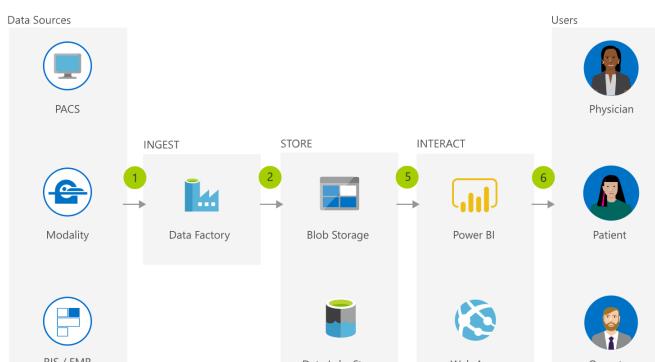
HIPAA and HITRUST compliant health data AI



HPC Media Rendering

11/04/2020 2 min read

Optimize the media rendering process with a step-by-step HPC solution architecture from Azure that combines Azure CycleCloud and HPC Cache.



Medical Data Storage Solutions

12/16/2019 2 min read

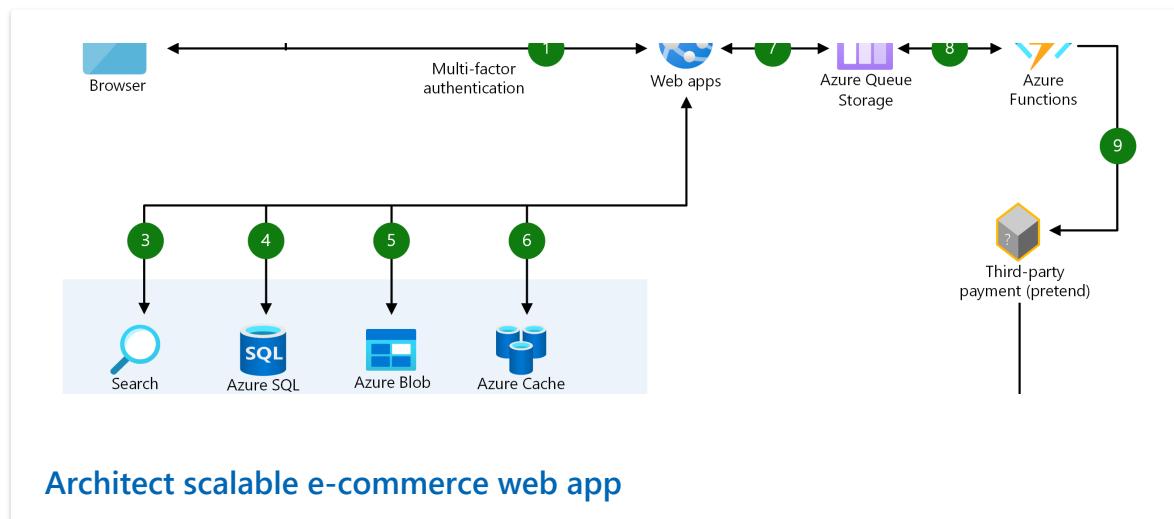
Store healthcare data effectively and affordably with cloud-based solutions from Azure. Manage medical records with the highest level of built-in security.

[view all](#)

Web applications

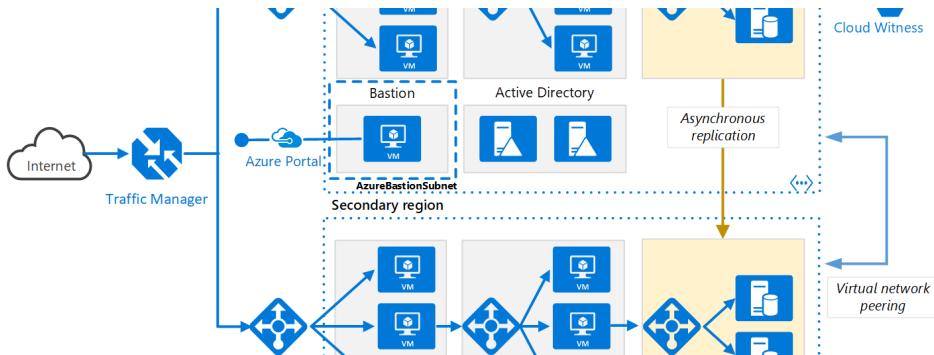
AWS service	Azure service	Description
Elastic Beanstalk ↗	App Service ↗	Managed hosting platform providing easy to use services for deploying and scaling web applications and services.
API Gateway ↗	API Management ↗	A turnkey solution for publishing APIs to external and internal consumers.
CloudFront ↗	Front Door ↗	Azure Front Door is a modern cloud content delivery network (CDN) service that delivers high performance, scalability, and secure user experiences for your content and applications.
Global Accelerator ↗	Front Door ↗	Easily join your distributed microservices architectures into a single global application using HTTP load balancing and path-based routing rules. Automate turning up new regions and scale-out with API-driven global actions, and independent fault-tolerance to your back end microservices in Azure-or anywhere.
Global Accelerator ↗	Cross-regional load balancer	Distribute and load balance traffic across multiple Azure regions via a single, static, global anycast public IP address.
LightSail ↗	App Service ↗	Build, deploy, and scale web apps on a fully managed platform.
App Runner ↗	Web App for Containers ↗	Easily deploy and run containerized web apps on Windows and Linux.
Amplify ↗	Static Web Apps ↗	Boost productivity with a tailored developer experience, CI/CD workflows to build and deploy your static content hosting, and dynamic scale for integrated serverless APIs.

Web architectures



12/16/2019 1 min read

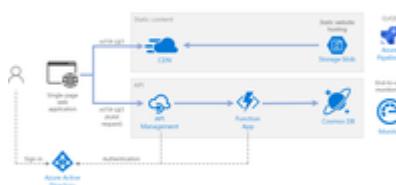
The e-commerce website includes simple order processing workflows with the help of Azure services. Using Azure Functions and Web Apps, developers can focus on building personalized experiences and let Azure take care of the infrastructure.



Multi-region N-tier application

6/18/2019 10 min read

Deploy an application on Azure virtual machines in multiple regions for high availability and resiliency.



Serverless web application

5/28/2019 16 min read

This reference architecture shows a serverless web application, which serves static content from Azure Blob Storage and implements an API using Azure Functions.

[view all](#)

End-user computing

AWS service	Azure service	Description
WorkSpaces ↗, AppStream 2.0 ↗	Azure Virtual Desktop	Manage virtual desktops and applications to enable corporate network and data access to users, anytime, anywhere, from supported devices. Amazon WorkSpaces support Windows and Linux virtual desktops. Azure Virtual Desktop supports multi-session Windows 10 virtual desktops.
WorkLink ↗	Application Proxy	Provides access to intranet applications, without requiring VPN connectivity. Amazon WorkLink is limited to iOS and Android devices.

Miscellaneous

Area	AWS service	Azure service	Description
Backend process logic	Step Functions ↗	Logic Apps ↗	Cloud technology to build distributed applications using out-of-the-box connectors to reduce integration challenges. Connect apps, data, and devices on-premises or in the cloud.
Enterprise application services	WorkMail ↗, WorkDocs ↗, Chime ↗	Microsoft 365 ↗	Fully integrated cloud service that provides communications, email, and document management in the cloud and is available on a wide variety of devices.
Gaming	GameLift ↗	PlayFab ↗	Managed services for hosting dedicated game servers.
Media transcoding	Elastic Transcoder ↗	Media Services ↗	Services that offer broadcast-quality video streaming services, including various transcoding technologies.
Workflow	Step Functions ↗	Logic Apps ↗	Serverless technology for connecting apps, data and devices anywhere, whether on-premises or in the cloud for large ecosystems of SaaS and cloud-based connectors.
Hybrid	Outposts ↗	Stack ↗	Azure Stack is a hybrid cloud platform that enables you to run Azure services in your company's or service provider's datacenter. As a developer, you can build apps on Azure Stack. You can then deploy them to either Azure Stack or Azure, or you can build truly hybrid apps that take advantage of connectivity between an Azure Stack cloud and Azure.

Area	AWS service	Azure service	Description
Media	Elemental MediaConvert	Media Services	Cloud-based media workflow platform to index, package, protect, and stream video at scale.
Satellite	Ground Station	Azure Orbital	Fully managed cloud-based ground station as a service.
Quantum computing	Amazon Braket	Azure Quantum	Managed quantum computing service that developers, researchers, and businesses can use to run quantum computing programs.

Next steps

If you are new to Azure, review the interactive [Core Cloud Services - Introduction to Azure](#) module.

Authenticate runbooks with Amazon Web Services

Article • 10/25/2022

You can automate common tasks with resources in Amazon Web Services (AWS) using Automation runbooks in Azure. You can automate many tasks in AWS using Automation runbooks similar to the resources in Azure. Ensure that you have the Azure subscription to authenticate.

Obtain AWS subscription and credentials

Ensure that you obtain an AWS subscription and specify a set of AWS credentials to authenticate your runbooks running from Azure Automation. Specific credentials required are the AWS Access Key and Secret Key. See [Using AWS Credentials](#).

Configure Automation account

You can use an existing Automation account to authenticate with AWS. Alternatively, you can dedicate an account for runbooks targeting AWS resources. In this case, create a new [Automation account](#).

Store AWS credentials

You must store the AWS credentials as assets in Azure Automation. See [Managing Access Keys for your AWS Account](#) for instructions on how to create the Access Key and the Secret Key. When the keys are available, copy the Access Key ID and the Secret Key ID in a safe place. You can download your key file to store it safely.

Create credential asset

After you have created and copied your AWS security keys, you must create a Credential asset with the Automation account. The asset allows you to securely store the AWS keys and reference them in your runbooks. See [Create a new credential asset with the Azure portal](#).

Enter the following AWS information in the fields provided:

- Name - **AWScred**, or an appropriate value following your naming standards

- **User name** - Your access ID
- **Password** - Name of your Secret Key

Next steps

- To learn how to create runbooks to automate tasks in AWS, see [Deploy an Amazon Web Services VM with a runbook](#).

Deploy an Amazon Web Services VM with a runbook

Article • 04/01/2021

In this article, you learn how you can leverage Azure Automation to provision a virtual machine in your Amazon Web Service (AWS) subscription and give that VM a specific name - which AWS refers to as "tagging" the VM.

Prerequisites

You need to have an Azure Automation account and an Amazon Web Services (AWS) subscription. For more information on setting up an Azure Automation account and configuring it with your AWS subscription credentials, review [Configure Authentication with Amazon Web Services](#). This account should be created or updated with your AWS subscription credentials before proceeding, as you reference this account in the sections below.

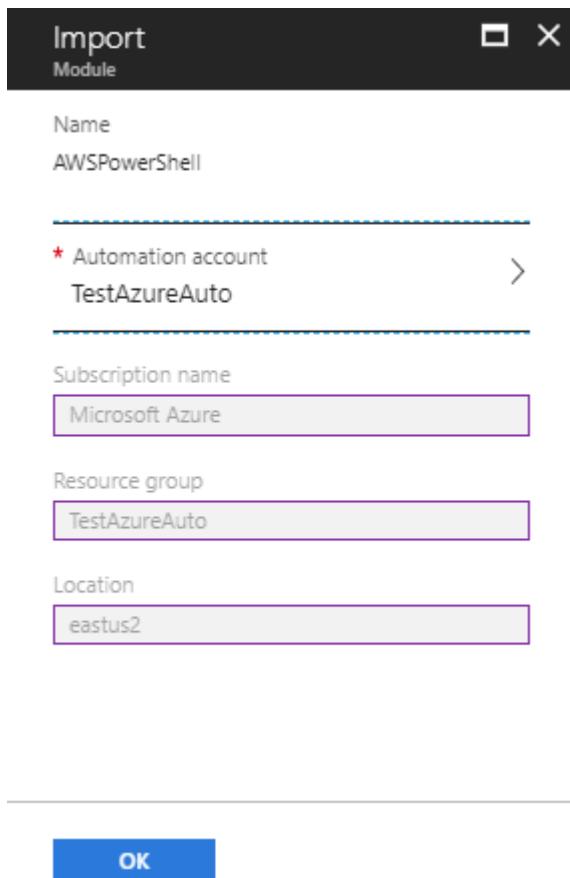
Deploy Amazon Web Services PowerShell Module

Your VM provisioning runbook uses the AWS PowerShell module to do its work. Use the following steps to add the module to your Automation account that is configured with your AWS subscription credentials.

1. Open your web browser and navigate to the [PowerShell Gallery](#) and click on the **Deploy to Azure Automation** button.

The screenshot shows the PowerShell Gallery website. At the top, there's a navigation bar with links for Home, Get Started, Modules (which is underlined), Scripts, Publish, and Statistics. There's also a search bar and a 'Register | Sign in' button. The main content area displays the 'AWS Tools for Windows PowerShell 3.1.58.0' module. It features a large blue icon with a white greater-than sign (>). Below the icon, it says '1,119 Downloads' and '3 Downloads of 3.1.58.0' (Last published: 2016-04-07). On the right, there are sections for 'Inspect' (with a command: PS> Save-Module -Name AWSPowerShell -Path <path>), 'Install' (with a command: PS> Install-Module -Name AWSPowerShell), and 'Deploy' (with a 'Deploy to Azure Automation' button).

2. You are taken to the Azure login page and after authenticating, you will be routed to the Azure portal and presented with the following page:



3. Select the Automation account to use and click OK to start deployment.

(!) Note

When Azure Automation imports a PowerShell module, it extracts the cmdlets. The activities don't appear until Automation has completely finished

importing the module and extracting the cmdlets. This process can take a few minutes.

4. In the Azure portal, open your Automation account.
5. Click on the **Assets** tile and, on the Assets pane, select **Modules**.
6. On the Modules page, you see the **AWSPowerShell** module in the list.

Create AWS deploy VM runbook

Once the AWS PowerShell Module has been deployed, you can now author a runbook to automate provisioning a virtual machine in AWS using a PowerShell script. The steps below demonstrate how to use native PowerShell script in Azure Automation.

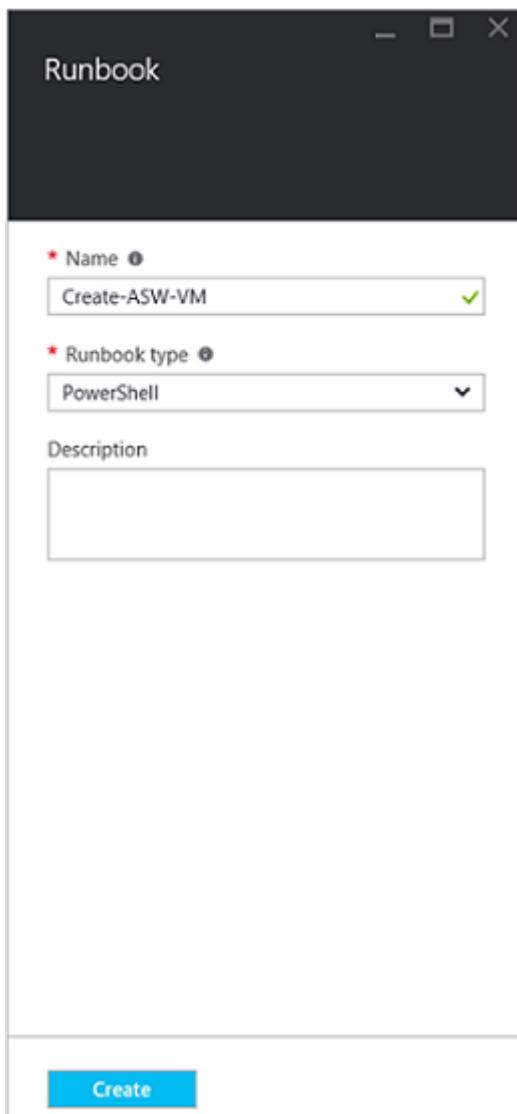
ⓘ Note

For further options and information regarding this script, please visit the [PowerShell Gallery](#).

1. Download the PowerShell script New-AwsVM from the PowerShell Gallery by opening a PowerShell session and typing the following command:

```
PowerShell  
  
Save-Script -Name New-AwsVM -Path <path>
```

2. From the Azure portal, open your Automation account and select **Runbooks** under **Process Automation**.
3. From the Runbooks page, select **Add a runbook**.
4. On the Add a runbook pane, select **Quick Create** to create a new runbook.
5. On the Runbook properties pane, type in a name for your runbook.
6. From the **Runbook type** drop-down list, select **PowerShell**, and then click **Create**.



- When the Edit PowerShell Runbook page appears, copy and paste the PowerShell script into the runbook authoring canvas.

```
34 #>
35 #ToDo:
36 #Change the default values for the following parameters if they do not match with yours:
37 #AWSRegion, EC2ImageName, MinCount, MaxCount, InstanceType
38 #Create an Azure Automation Asset called "AwsCred"
39 #Turn on Log verbose records and optionally Log progress records under the runbook settings to see verbose
40
41 param (
42     [Parameter(Mandatory=$true)]
43     [string]$Vmname,
44     [ValidateNotNullOrEmpty()]
45     [string]$AWSRegion = "us-west-2",
46     [ValidateNotNullOrEmpty()]
47     [string]$EC2ImageName = "WINDOWS_2012R2_BASE",
48     [ValidateNotNullOrEmpty()]
49     [string]$MinCount = 1,
50     [ValidateNotNullOrEmpty()]
51     [string]$MaxCount = 1,
52     [ValidateNotNullOrEmpty()]
53     [string]$InstanceType = "t2.micro"
54 )
55
56 # Get credentials to authenticate against AWS
57 $AwsCred = Get-AutomationPSCredential -Name "AwsCred"
58 $AwsAccessKeyId = $AwsCred.UserName
59 $AwsSecretKey = $AwsCred.GetNetworkCredential().Password
```

Note the following when working with the example PowerShell script:

- The runbook contains a number of default parameter values. Evaluate all default values and update where necessary.
- If you have stored your AWS credentials as a credential asset named differently from `AWScred`, you need to update the script on line 57 to match accordingly.
- When working with the AWS CLI commands in PowerShell, especially with this example runbook, you must specify the AWS region. Otherwise, the cmdlets fail. View AWS topic [Specify AWS Region](#) in the AWS Tools for PowerShell document for further details.

8. To retrieve a list of image names from your AWS subscription, launch PowerShell ISE and import the AWS PowerShell Module. Authenticate against AWS by replacing `Get-AutomationPSCredential` in your ISE environment with `AWScred = Get-Credential`. This statement prompts for your credentials and you can provide your access key ID for the user name and your secret access key for the password.

PowerShell

```
#Sample to get the AWS VM available images
#Please provide the path where you have downloaded the AWS PowerShell
module
Import-Module AWSPowerShell
$AwsRegion = "us-west-2"
$AwsCred = Get-Credential
$AwsAccessKeyId = $AwsCred.UserName
$AwsSecretKey = $AwsCred.GetNetworkCredential().Password

# Set up the environment to access AWS
Set-AwsCredentials -AccessKey $AwsAccessKeyId -SecretKey
$AwsSecretKey -StoreAs AWSProfile
Set-DefaultAWSRegion -Region $AwsRegion

Get-EC2ImageByName -ProfileName AWSProfile
```

The following output is returned:

```
PS C:\> Get-EC2ImageByName -ProfileName AwsProfile
WINDOWS_2012R2_BASE
WINDOWS_2012R2_SQL_SERVER_EXPRESS_2014
WINDOWS_2012R2_SQL_SERVER_STANDARD_2014
WINDOWS_2012R2_SQL_SERVER_WEB_2014
WINDOWS_2012_BASE
WINDOWS_2012_SQL_SERVER_EXPRESS_2014
WINDOWS_2012_SQL_SERVER_STANDARD_2014
WINDOWS_2012_SQL_SERVER_WEB_2014
WINDOWS_2012_SQL_SERVER_EXPRESS_2012
WINDOWS_2012_SQL_SERVER_STANDARD_2012
WINDOWS_2012_SQL_SERVER_WEB_2012
WINDOWS_2012_SQL_SERVER_EXPRESS_2008
WINDOWS_2012_SQL_SERVER_STANDARD_2008
WINDOWS_2012_SQL_SERVER_WEB_2008
WINDOWS_2008R2_BASE
WINDOWS_2008R2_SQL_SERVER_EXPRESS_2012
WINDOWS_2008R2_SQL_SERVER_STANDARD_2012
WINDOWS_2008R2_SQL_SERVER_WEB_2012
WINDOWS_2008R2_SQL_SERVER_EXPRESS_2008
WINDOWS_2008R2_SQL_SERVER_STANDARD_2008
WINDOWS_2008R2_SQL_SERVER_WEB_2008
WINDOWS_2008RTM_BASE
WINDOWS_2008RTM_SQL_SERVER_EXPRESS_2008
WINDOWS_2008RTM_SQL_SERVER_STANDARD_2008
WINDOWS_BEANSTALK_IIS75
WINDOWS_2012_BEANSTALK_IIS8
VPC_NAT
```

9. Copy and paste the one of the image names in an Automation variable as referenced in the runbook as `$InstanceType`. Since, in this example, you are using the free AWS tiered subscription, you use `t2.micro` for your runbook example.
10. Save the runbook, then click **Publish** to publish the runbook and then **Yes** when prompted.

Test the AWS VM runbook

1. Verify that the runbook creates an asset called `Awscred` for authenticating against AWS, or update the script to reference your credential asset name.
2. Verify your new runbook and make sure that all parameter values have been updated has necessary. Ensure that the AWS PowerShell module has been imported into Azure Automation.
3. In Azure Automation, set **Log verbose records** and optionally **Log progress records** under the runbook operation **Logging and tracing** to **On**.



<<

Save

Discard

Webhooks

RUNBOOK SETTINGS

Properties

Description

Logging and tracing

SETTINGS

Locks

Automation script

Logging

Log verbose records

Log progress records

Activity-level tracing

This configuration is available only for graphical runbooks.

Trace level

4. Click **Start** to start the runbook, then click **OK** when the Start Runbook pane opens.
5. On the Start Runbook pane, provide a VM name. Accept the default values for the other parameters that you preconfigured in the script. Click **OK** to start the runbook job.



Parameters

★ VMNAME ⓘ

Enter a value

Mandatory, String

AWSREGION ⓘ

Default will be used

Optional, String, Default: "us-west-2"

EC2IMAGENAME ⓘ

Default will be used

Optional, String, Default:
"WINDOWS_2012R2_BASE"

MINCOUNT ⓘ

Default will be used

Optional, String, Default: 1

MAXCOUNT ⓘ

Default will be used

Optional, String, Default: 1

INSTANCETYPE ⓘ

Default will be used

Optional, String, Default: "t2.micro"

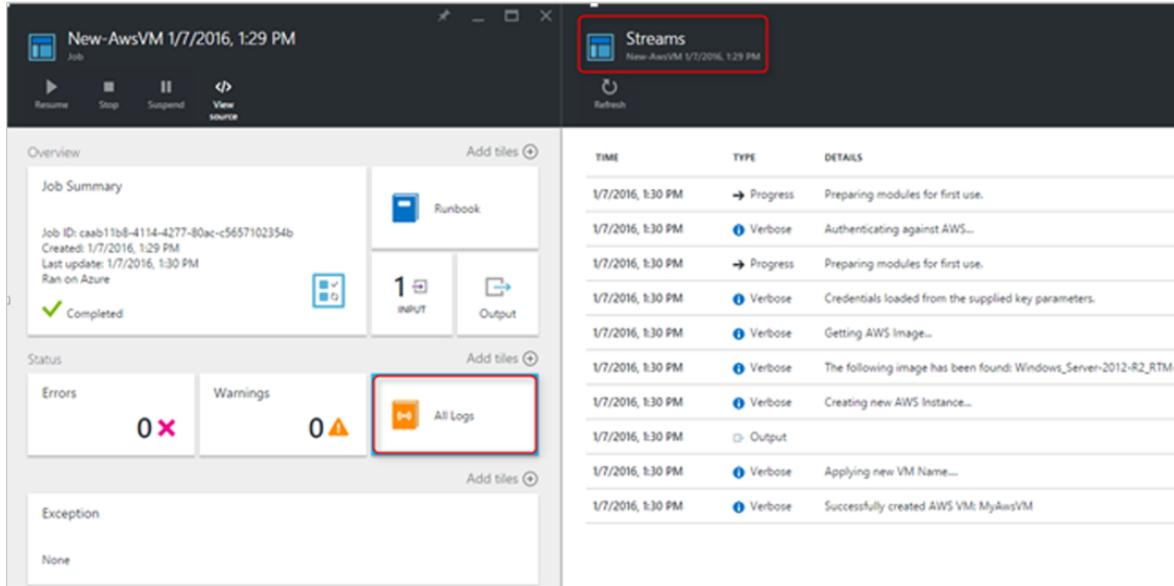
Run Settings

Run on ⓘ

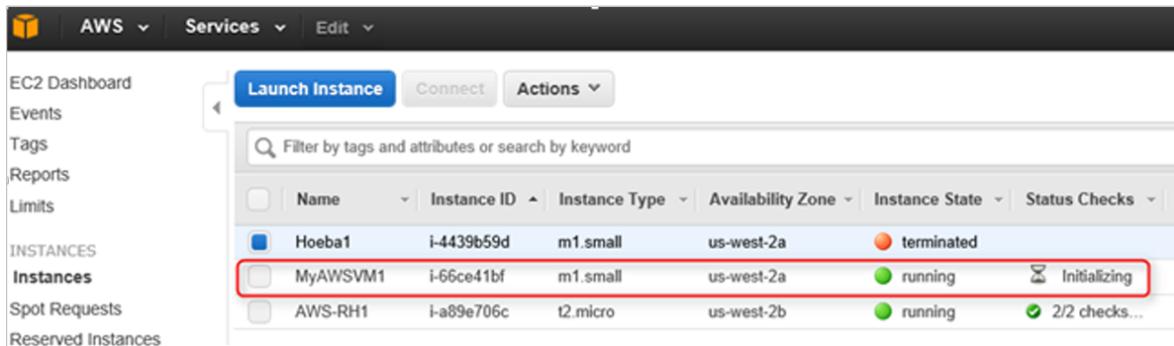
Azure Hybrid Worker

OK

6. A Job pane is opened for the runbook job that you created. Close this pane.
7. You can view progress of the job and view output streams by selecting **All Logs** from the runbook Job pane.



8. To confirm that the VM is being provisioned, log into the AWS Management Console if you aren't currently logged in.



Next steps

- To find out what runbooks are supported, see [Azure Automation runbook types](#).
- To work with runbooks, see [Manage runbooks in Azure Automation](#).
- For details of PowerShell, see [PowerShell Docs](#).
- For script support, see [Native PowerShell script support in Azure Automation](#).
- For a PowerShell cmdlet reference, see [Az.Automation](#).

Set up and configure AWS Cost and Usage report integration

Article • 11/30/2023

With Amazon Web Services (AWS) Cost and Usage report (CUR) integration, you monitor and control your AWS spending in Cost Management. The integration allows a single location in the Azure portal where you monitor and control spending for both Azure and AWS. This article explains how to set up the integration and configure it so that you can use Cost Management features to analyze costs and review budgets.

Cost Management processes the AWS Cost and Usage report stored in an S3 bucket by using your AWS access credentials to get report definitions and download report GZIP CSV files.

Create a Cost and Usage report in AWS

Using a Cost and Usage report is the AWS-recommended way to collect and process AWS costs. The Cost Management cross cloud connector supports cost and usage reports configured at the management (consolidated) account level. For more information, see the [AWS Cost and Usage Report](#) documentation.

Use the **Cost & Usage Reports** page of the Billing and Cost Management console in AWS to create a Cost and Usage report with the following steps:

1. Sign in to the AWS Management Console and open the [Billing and Cost Management console](#).
2. In the navigation pane, select **Cost & Usage Reports**.
3. Select **Create report**.
4. For **Report name**, enter a name for your report.
5. Under **Additional report details**, select **Include resource IDs**.
6. For **Data refresh settings**, select whether you want the AWS Cost and Usage report to refresh if AWS applies refunds, credits, or support fees to your account after finalizing your bill. When a report refreshes, a new report is uploaded to Amazon S3. We recommend that you leave the setting selected.
7. Select **Next**.
8. For **S3 bucket**, choose **Configure**.
9. In the Configure S3 Bucket dialog box, enter a bucket name and the Region where you want to create a new bucket and choose **Next**.
10. Select **I have confirmed that this policy is correct**, then select **Save**.

11. (Optional) For Report path prefix, enter the report path prefix that you want prepended to the name of your report.
If skipped, the default prefix is the name that you specified for the report. The date range has the `/report-name/date-range/` format.
12. For **Time unit**, choose **Hourly**.
13. For **Report versioning**, choose whether you want each version of the report to overwrite the previous version, or if you want more new reports.
14. For **Enable data integration for**, no selection is required.
15. For **Compression**, select **GZIP**.
16. Select **Next**.
17. After you review the settings for your report, select **Review and Complete**.
Note the report name. You use it in later steps.

It can take up to 24 hours for AWS to start delivering reports to your Amazon S3 bucket. After delivery starts, AWS updates the AWS Cost and Usage report files at least once a day. You can continue configuring your AWS environment without waiting for delivery to start.

 **Note**

Cost and usage reports configured at the member (linked) account level aren't currently supported.

Create a policy and role in AWS

Cost Management accesses the S3 bucket where the Cost and Usage report is located several times a day. The service needs access to credentials to check for new data. You create a role and policy in AWS to allow Cost Management to access it.

To enable role-based access to an AWS account in Cost Management, the role is created in the AWS console. You need to have the *role ARN* and *external ID* from the AWS console. Later, you use them on the **Create an AWS connector** page in Cost Management.

Use the Create Policy wizard

1. Sign in into your AWS console and select **Services**.
2. In the list of services, select **IAM**.
3. Select **Policies**.
4. Select **Create policy**.

5. Select **Choose a service**.

Configure permission for the Cost and Usage report

1. Enter **Cost and Usage Report**.
2. Select **Access level > Read > DescribeReportDefinitions**. This step allows Cost Management to read what CUR reports are defined and determine if they match the report definition prerequisite.
3. Select **Add more permissions**.

Configure permission for your S3 bucket and objects

1. Select **Choose a service**.
2. Enter **S3**.
3. Select **Access level > List > ListBucket**. This action gets the list of objects in the S3 Bucket.
4. Select **Access level > Read > GetObject**. This action allows the download of billing files.
5. Select **Resources > Specific**.
6. In **bucket**, select the **Add ARNs** link to open another window.
7. In **Resource Bucket name**, enter the bucket used to store the CUR files.
8. Select **Add ARNs**.
9. In **object**, select **Any**.
10. Select **Add more permissions**.

Configure permission for Cost Explorer

1. Select **Choose a service**.
2. Enter **Cost Explorer Service**.
3. Select **All Cost Explorer Service actions (ce:*)**. This action validates that the collection is correct.
4. Select **Add more permissions**.

Add permission for AWS Organizations

1. Enter **Organizations**.
2. Select **Access level > List > ListAccounts**. This action gets the names of the accounts.
3. Select **Add more permissions**.

Configure permissions for Policies

1. Enter IAM.
2. Select Access level > List > **ListAttachedRolePolicies** and **ListPolicyVersions** and **ListRoles**.
3. Select Access level > Read > **GetPolicyVersion**.
4. Select **Resources** > policy, and then select **Any**. These actions allow verification that only the minimal required set of permissions were granted to the connector.
5. Select **Next**.

Review and create

1. In Review Policy, enter a name for the new policy. Verify that you entered the correct information.
2. Add tags. You can enter tags you wish to use or skip this step. This step isn't required to create a connector in Cost Management.
3. Select **Create policy** to complete this procedure.

The policy JSON should resemble the following example. Replace `bucketname` with the name of your S3 bucket, `accountname` with your account number and `rolename` with the role name you created.

JSON

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "organizations>ListAccounts",  
                "iam>ListRoles",  
                "ce:*",  
                "cur>DescribeReportDefinitions"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "VisualEditor1",  
            "Effect": "Allow",  
            "Action": [  
                "s3>GetObject",  
                "s3>ListBucket",  
                "iam>GetPolicyVersion",  
                "iam>ListPolicyVersions",  
                "iam>ListAttachedRolePolicies"  
            ]  
        }  
    ]  
}
```

```
        ],
        "Resource": [
            "arn:aws:s3:::bucketname",
            "arn:aws:s3:::bucketname/*",
            "arn:aws:iam::accountnumber:policy/*",
            "arn:aws:iam::accountnumber:role/rolename"
        ]
    }
}
```

Use the Create a New Role wizard

1. Sign in to your AWS console and select **Services**.
2. In the list of services, select **IAM**.
3. Select **Roles** and then select **Create Role**.
4. On the **Select trusted entity** page, select **AWS account** and then under **An AWS account**, select **Another AWS account**.
5. Under **Account ID**, enter **432263259397**.
6. Under **Options**, select **Require external ID** (**Best practice when a third party will assume this role**).
7. Under **External ID**, enter the external ID, which is a shared passcode between the AWS role and Cost Management. Note the external ID, because you use it on the **New Connector** page in Cost Management. Microsoft recommends that you use a strong passcode policy when entering the external ID. The external ID should comply with AWS restrictions:
 - Type: String
 - Length constraints: Minimum length of 2. Maximum length of 1224.
 - Must satisfy regular expression pattern: `[\w+=,.@:/-]*`

 **Note**

Don't change the selection for **Require MFA**. It should remain cleared.

8. Select **Next**.
9. On the search bar, search for your new policy and select it.
10. Select **Next**.
11. In **Role details**, enter a role name. Verify that you entered the correct information.
Note the name entered because you use it later when you set up the Cost Management connector.
12. Optionally, add tags. You can enter any tags like or skip this step. This step isn't required to create a connector in Cost Management.

13. Select **Create role**.

Set up a new connector for AWS in Azure

Use the following information to create an AWS connector and start monitoring your AWS costs.

Note

The Connector for AWS remains active after the trial period ends if you set the auto-renew configuration to **On** during the initial setup. Otherwise, the connector is disabled following its trial. It may remain disabled for three months before it's permanently deleted. After the connector is deleted, the same connection can't be reactivated. For assistance with a disabled connector or to create a new connection after it's deleted, create a [support request in the Azure portal](#).

Prerequisites

- Ensure you have at least one management group enabled. A management group is required to link your subscription to the AWS service. For more information about creating a management group, see [Create a management group in Azure](#).
- Ensure that you're an administrator of the subscription.
- Complete the setup required for a new AWS connector, as described in the [Create a Cost and Usage report in AWS](#) section.

Create a new connector

1. Sign in to the [Azure portal](#).
2. Navigate to **Cost Management + Billing** and select a billing scope, if necessary.
3. Select **Cost analysis** and then select **Settings**.
4. Select **Connectors for AWS**.
5. Select **Add connector**.

6. On the **Create connector** page, in **Display name**, enter a name for your connector.

Create connector

Create a connector to bring AWS cost and usage report data into Azure Cost Management. After you create a connector, you'll have a single location to view all your cloud costs.

Basics AWS properties Review

Display name *

 ✓

Management Group

Please select a management group

[Create new](#) | [Refresh](#)

i Management groups allow you to group your Azure subscriptions and AWS linked accounts. Select a management group to view all Azure and AWS costs together.
[Learn more](#)

Billing

i You will receive 90 days of free trial at the time of creating a connector. After this period, you will be charged 1% of your monthly AWS costs under cost management.
Please select a subscription for us to charge.
[Learn more](#)

Subscription * i

Please select a subscription

Auto-Renew

On Off

7. Optionally, select the default management group. It stores all discovered linked accounts. You can set it up later.
8. In the **Billing** section, select **Auto-Renew** to **On** if you want to ensure continuous operation. If you select the automatic option, you must select a billing subscription.
9. For **Role ARN**, enter the value that you used when you set up the role in AWS.
10. For **External ID**, enter the value that you used when you set up the role in AWS.
11. For **Report Name**, enter the name that you created in AWS.
12. Select **Next** and then select **Create**.

It might take a few hours for the new AWS scopes, AWS consolidated account, AWS linked accounts, and their cost data to appear.

After you create the connector, we recommend that you assign access control to it. Users are assigned permissions to the newly discovered scopes: AWS consolidated account and AWS linked accounts. The user who creates the connector is the owner of the connector, the consolidated account, and all linked accounts.

Assigning connector permissions to users after discovery occurs doesn't assign permissions to the existing AWS scopes. Instead, only new linked accounts are assigned permissions.

Take other steps

- [Set up management groups](#), if you haven't already.
- Check that new scopes are added to your scope picker. Select **Refresh** to view the latest data.
- On the **Cloud connectors** page, select your connector and select **Go to billing account** to assign the linked account to management groups.

Note

Management groups aren't currently supported for Microsoft Customer Agreement (MCA) customers. MCA customers can create the connector and view their AWS data. However, MCA customers can't view their Azure costs and AWS costs together under a management group.

Manage AWS connectors

When you select a connector on the **Connectors for AWS** page, you can:

- Select **Go to Billing Account** to view information for the AWS consolidated account.
- Select **Access Control** to manage the role assignment for the connector.
- Select **Edit** to update the connector. You can't change the AWS account number, because it appears in the role ARN. But you can create a new connector.
- Select **Verify** to rerun the verification test to make sure that Cost Management can collect data by using the connector settings.

<ConnectorName>

Connector for AWS


 Refresh
 Access Control
 Verify
 Edit
 Delete

Essentials

[View as JSON](#)

Consolidated account name	Billing
Cloudyn Software Ltd.	Trial
Consolidated account ID	Auto-Renew
<ConsolidatedAccountID>	On
Consolidated account number	Billing subscription
432263259397	11111111-1111-1111-1111-111111111111
Management group	Current service period
Trey Research	Aug 01, 2020 to Aug 31, 2020
Report name	Next renewal
NewReportWithResources	Sep 01, 2020

Data collection

Collection status	: Active
Last synced	: 8/28/2020, 1:32:46 AM
Last updated by AWS	: 8/28/2020, 12:40:35 AM

Set up Azure management groups

Place your Azure subscriptions and AWS linked accounts in the same management group to create a single location where you can see cross-cloud provider information. If you want to configure your Azure environment with management groups, see [Initial setup of management groups](#).

If you want to separate costs, you can create a management group that holds just AWS linked accounts.

Set up an AWS consolidated account

The AWS consolidated account combines billing and payment for multiple AWS accounts. It also acts as an AWS linked account. You can view the details for your AWS consolidated account using the link on the AWS connector page.

The screenshot shows the AWS CloudWatch Metrics Insights interface. At the top, it displays the account name "Cloudyn Software Ltd." and the status "AWS Consolidated Account". Below this are navigation links for "Refresh", "Access Control", and "Change Management Group". A sidebar on the left shows "Essentials" information: Account number (432263259397), Account ID (aws-432263259397), and Total linked accounts (1). To the right, there's a section for "Current month's cost" (\$100.9K) and "Last synced" (8/31/2020, 11:20:00 AM). A "View as JSON" link is also present. Below this is a search bar with the placeholder "Filter linked accounts by name, account number or management group...". A table follows, showing a single linked account: "Cloudyn Software Ltd." with account number 432263259397, management group "Trey Research", and current month's cost "\$100.4K".

From the page, you can:

- Select **Update** to bulk update the association of AWS linked accounts with a management group.
- Select **Access Control** to set the role assignment for the scope.

Permissions for an AWS consolidated account

By default, permissions for an AWS consolidated account are set upon the account's creation, based on the AWS connector permissions. The connector creator is the owner.

You manage the access level by using the **Access Level** page of the AWS consolidated account. However, AWS linked accounts don't inherit permissions to the AWS consolidated account.

Set up an AWS linked account

The AWS linked account is where AWS resources are created and managed. A linked account also acts as a security boundary.

From this page, you can:

- Select **Update** to update the association of an AWS linked account with a management group.
- Select **Access Control** to set a role assignment for the scope.

Cloudyn Software Ltd.

AWS Linked Account



Refresh Access Control Change Management Group

Essentials

[View as JSON](#)

Account number	Current month's cost
432263259397	\$100.4K
Management group	Last synced
Trey Research	8/31/2020, 11:20:00 AM
Consolidated account number	Last updated by AWS
432263259397	8/31/2020, 10:03:43 AM

Permissions for an AWS linked account

By default, permissions for an AWS linked account are set upon creation, based on the AWS connector permissions. The connector creator is the owner. You manage the access level by using the **Access Level** page of the AWS linked account. AWS linked accounts don't inherit permissions from an AWS consolidated account.

AWS linked accounts always inherit permissions from the management group that they belong to.

Next steps

- Now that you set up and configured AWS Cost and Usage report integration, continue to [Manage AWS costs and usage](#).
- If you're unfamiliar with cost analysis, see [Explore and analyze costs with cost analysis](#) quickstart.
- If you're unfamiliar with budgets in Azure, see [Create and manage budgets](#).

Manage AWS costs and usage in Azure

Article • 11/30/2023

After you've set up and configured AWS Cost and Usage report integration for Cost Management, you're ready to start managing your AWS costs and usage. This article helps you understand how to use cost analysis and budgets in Cost Management to manage your AWS costs and usage.

If you haven't already configured the integration, see [Set up and configure AWS Usage report integration](#).

Before you begin: If you're unfamiliar with cost analysis, see the [Explore and analyze costs with Cost analysis](#) quickstart. And, if you're unfamiliar with budgets in Azure, see the [Create and manage budgets](#) tutorial.

View AWS costs in cost analysis

AWS costs are available in Cost Analysis in the following scopes:

- AWS linked accounts under a management group
- AWS linked account costs
- AWS consolidated account costs

The next sections describe how to use the scopes so that you see cost and usage data for each one.

View AWS linked accounts under a management group

Viewing costs by using the management group scope is the only way to see aggregated costs coming from different Azure subscriptions and AWS linked accounts. Using a management group provides a cross-cloud view to view costs from Azure and AWS together.

In cost analysis, open the scope picker and select the management group that holds your AWS linked accounts. Here's an example image in the Azure portal:

Select scope

X

Cost Management + Billing

Scopes are levels in the resource hierarchy where you manage and control access to one or more resources. Select a scope to see a filtered roll-up of all resources, products, and services.

[Learn more](#)

Root management g...



[Select this management group](#)

Search to filter items...

test mgt group

Demo mgt group > ⓘ

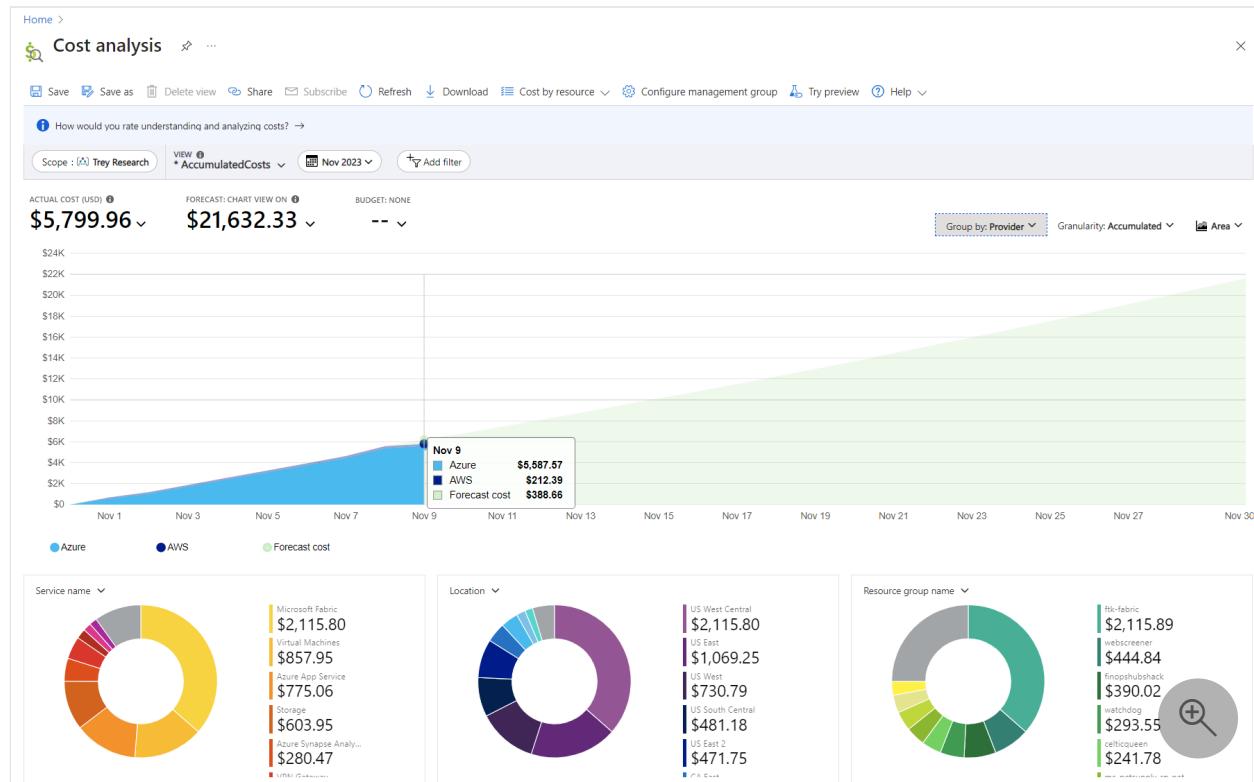
Cloud + AI Platform ⓘ

CnAI Orchestration Service Public Corp prod ⓘ

Trey Research

Cost Management Research

Here's an example showing the management group cost in cost analysis, grouped by Provider (Azure and AWS).



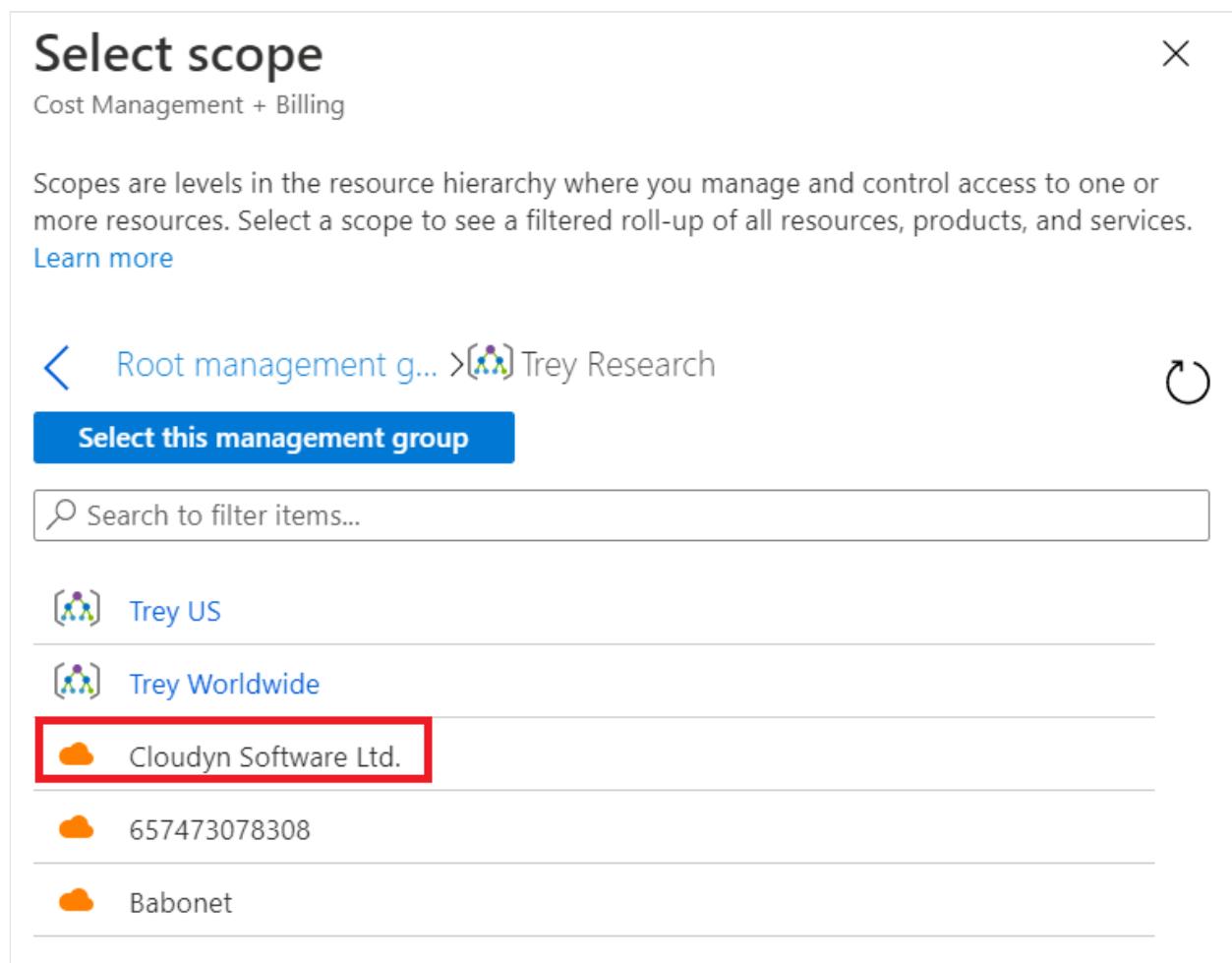
Note

Management groups aren't currently supported for Microsoft Customer Agreement (MCA) customers. MCA customers can create the connector and view their AWS data. However, MCA customers can't view their Azure costs and AWS costs together under a management group.

View AWS linked account costs

To view AWS link account costs, open the scope picker and select the AWS linked account. Note that linked accounts are associated to a management group, as defined in the AWS connector.

Here's an example that shows selecting an AWS linked account scope.



The screenshot shows the 'Select scope' dialog box from the Cost Management + Billing service. The title is 'Select scope' and there is a close button 'X'. Below the title, it says 'Cost Management + Billing'. A descriptive text states: 'Scopes are levels in the resource hierarchy where you manage and control access to one or more resources. Select a scope to see a filtered roll-up of all resources, products, and services.' There is a 'Learn more' link. At the top, there are navigation arrows: '< Root management g...' and '> Trey Research'. A blue button labeled 'Select this management group' is visible. Below the button is a search bar with the placeholder 'Search to filter items...'. A list of management groups is shown: 'Trey US', 'Trey Worldwide', 'Cloudyn Software Ltd.' (which is highlighted with a red box), '657473078308', and 'Babonet'.

View AWS consolidated account costs

To view AWS consolidated account costs, open the scope picker and select the AWS consolidated account. Here's an example that shows selecting an AWS consolidated account scope.

Select scope

X

Cost Management + Billing

Scopes are levels in the resource hierarchy where you manage and control access to one or more resources. Select a scope to see a filtered roll-up of all resources, products, and services.

[Learn more](#)

Microsoft



Current directory – [Switch directories](#)

Search to filter items...



<TestSubscriptionName>



<TestSubscriptionName>



<TestSubscriptionName>



<TestSubscriptionName>



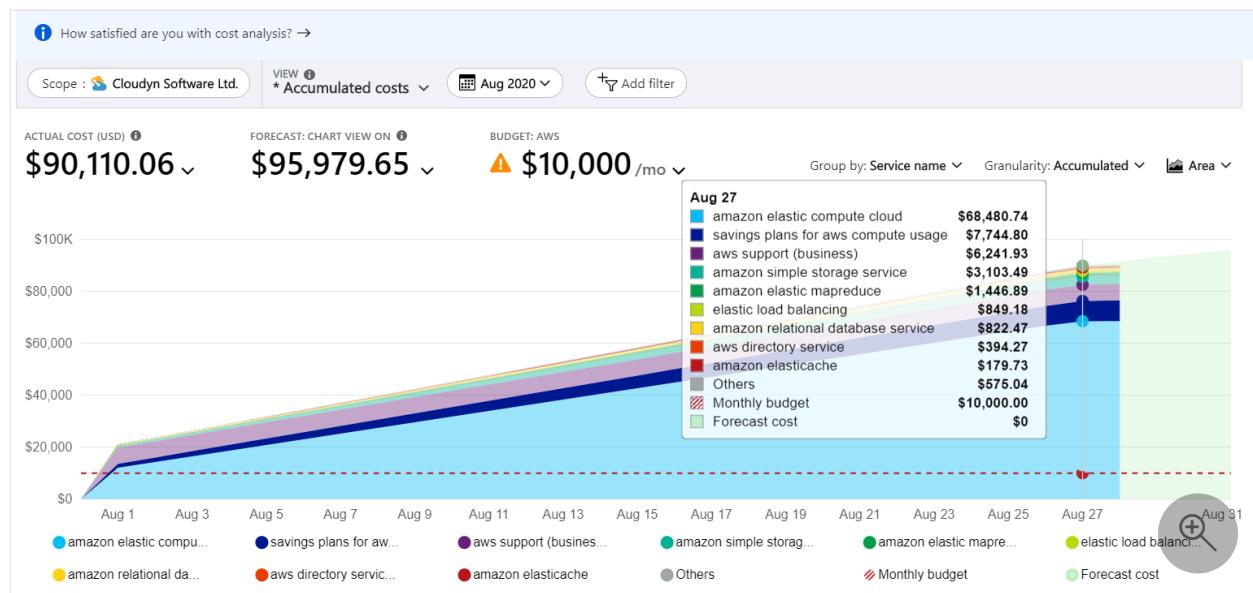
Cloudyn Software Ltd.

Only show subscriptions selected in the global filter. [Change filter](#)

[Select](#)

[Cancel](#)

This scope provides an aggregated view of all AWS linked accounts associated with the AWS consolidated account. Here's an example showing costs for an AWS consolidated account, grouped by service name.



Dimensions available for filtering and grouping

The following table describes dimensions available to group and filter by in cost analysis.

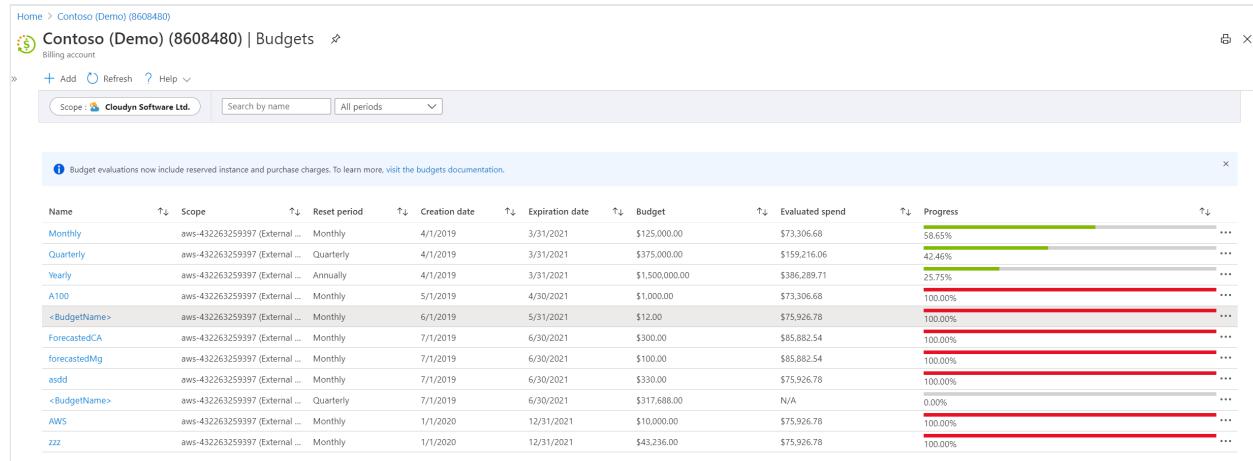
 Expand table

Dimension	Amazon CUR header	Scopes	Comments
Availability zone	lineitem/AvailabilityZone	All	
Location	product/Region	All	
Meter		All	
Meter category	lineItem/ProductCode	All	
Meter subcategory	lineitem/UsageType	All	
Operation	lineItem/Operation	All	
Resource	lineItem/ResourceId	All	
Resource type	product/instanceType	All	If product/instanceType is null, lineItem/UsageType is used.
ResourceGuid	N/A	All	Azure meter GUID.
Service name	product/ProductName	All	If product/ProductName is null, lineItem/ProductCode is used.
Service tier			
Subscription ID	lineItem/UsageAccountId	Consolidated account and management group	
Subscription name	N/A	Consolidated account and management group	Account names are collected using the AWS Organization API.
Tag	resourceTags	All	The <i>user:</i> prefix is removed from user-defined tags to allow cross-cloud tags. The <i>aws:</i> prefix is left intact.

Dimension	Amazon CUR header	Scopes	Comments
Billing account ID	bill/PayerAccountId	Management group	
Billing account name	N/A	Management group	Account names are collected using the AWS Organization API.
Provider	N/A	Management group	Either AWS or Azure.

Set budgets on AWS scopes

Use budgets to proactively manage costs and drive accountability in your organization. Budgets are set on the AWS consolidated account and AWS linked account scopes. Here's an example of budgets for an AWS consolidated account shown in Cost Management:



AWS data collection process

After setting up the AWS connector, data collection and discovery processes start. It might take few hours to collect all usage data. The duration depends on:

- The time needed to process the CUR files that are in the AWS S3 bucket.
- The time needed to create the AWS Consolidated account and AWS Linked account scopes.
- The time and frequency of AWS are writing the Cost and Usage Report files in the S3 bucket

AWS integration pricing

Each AWS connector gets 90 free trial days.

The list price is 1% of your AWS monthly costs. Each month you are charged based on your invoiced costs from the previous month.

Accessing AWS APIs may incur additional costs on AWS.

AWS integration limitations

- Budgets in Cost Management don't support management groups with multiple currencies. Management groups with multiple currencies won't see a budget evaluation. An error message is shown if you select a management group that has multiple currencies when you create a budget.
- Cloud connectors don't support AWS GovCloud (US), AWS Gov, or AWS China.
- Cost Management shows AWS *usage costs* only. Tax, support, refunds, RI, credits or any other charge types aren't supported yet.

Troubleshooting AWS integration

Use the following troubleshooting information to resolve common problems.

No permission to AWS Linked accounts

Error code: *Unauthorized*

There are two ways to get permissions to access AWS linked accounts costs:

- Get access to the management group that has the AWS Linked accounts.
- Have someone give you permission to the AWS linked account.

By default, the AWS connector creator is the owner of all the objects that the connector created. Including, the AWS consolidated account and the AWS linked account.

In order to be able to Verify the connector settings you will need at least a contributor role because a reader can't Verify connector settings

Collection failed with AssumeRole

Error code: *FailedToAssumeRole*

This error means that Cost Management is unable to call the AWS AssumeRole API. This problem can happen because of an issue with the role definition. Verify that the

following conditions are true:

- The external ID is the same as the one in the role definition and the connector definition.
- The role type is set to **Another AWS account Belonging to you or 3rd party**.
- The **Require MFA** choice is cleared.
- The trusted AWS account in the AWS Role is 432263259397.

Collection failed with Access Denied - CUR report definitions

Error code: *AccessDeniedReportDefinitions*

This error means that Cost Management is unable to see the Cost and Usage report definitions. This permission is used to validate that the CUR is defined as expected by Cost Management. See [Create a Cost and Usage report in AWS](#).

Collection failed with Access Denied - List reports

Error code: *AccessDeniedListReports*

This error means that Cost Management is unable to list the object in the S3 bucket where the CUR is located. AWS IAM policy requires a permission on the bucket and on the objects in the bucket. See [Create a role and policy in AWS](#).

Collection failed with Access Denied - Download report

Error code: *AccessDeniedDownloadReport*

This error means that Cost Management is unable to access and download the CUR files stored in the Amazon S3 bucket. Make sure that the AWS JSON policy attached to the role resembles the example shown at the bottom of the [Create a role and policy in AWS](#) section.

Collection failed since we did not find the Cost and Usage Report

Error code: *FailedToFindReport*

This error means that Cost Management can't find the Cost and Usage report that was defined in the connector. Make sure it isn't deleted and that the AWS JSON policy

attached to the role resembles the example shown at the bottom of the [Create a role and policy in AWS](#) section.

Unable to create or verify connector due to Cost and Usage Report definitions mismatch

Error code: *ReportIsNotValid*

This error relates to the definition of AWS Cost and Usage Report, we require specific settings for this report, see the requirements in [Create a Cost and Usage report in AWS](#).

Internal error when creating connector

Error code: *Create connector - Failed to create connector <ConnectorName>. Reason: Internal error. Please verify that the correct AWS properties were provided.*

This error can occur when your AWS connector and subscription are in different management groups. The AWS connector and subscription need to be in the same management group.

Next steps

- If you haven't already configured your Azure environment with management groups, see [Initial setup of management groups](#).

Use a Terraform plan to deploy an Amazon Web Services Amazon Elastic Compute Cloud instance and connect it to Azure Arc

Article • 03/31/2023

This article provides guidance for using the provided [Terraform](#) plan to deploy an Amazon Web Services (AWS) Amazon Elastic Compute Cloud (EC2) instance and connect it as an Azure Arc-enabled server resource.

Prerequisites

1. Clone the Azure Arc Jumpstart repository.

Console

```
git clone https://github.com/microsoft/azure_arc.git
```

2. [Install or update Azure CLI to version 2.7 and above](#). Use the following command to check your current installed version.

Console

```
az --version
```

3. [Generate SSH key](#) (or use existing SSH key)

4. [Create free AWS account](#)

5. [Install Terraform >= 0.12](#)

6. Create an Azure service principal.

To connect the AWS virtual machine to Azure Arc, an Azure service principal assigned with the Contributor role is required. To create it, sign in to your Azure account and run the following command. You can also run this command in [Azure Cloud Shell](#).

Console

```
az login  
az ad sp create-for-rbac -n "<Unique SP Name>" --role contributor
```

For example:

Console

```
az ad sp create-for-rbac -n "http://AzureArcAWS" --role contributor
```

Output should look like this:

JSON

```
{  
  "appId": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",  
  "displayName": "AzureArcAWS",  
  "name": "http://AzureArcAWS",  
  "password": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",  
  "tenant": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"  
}
```

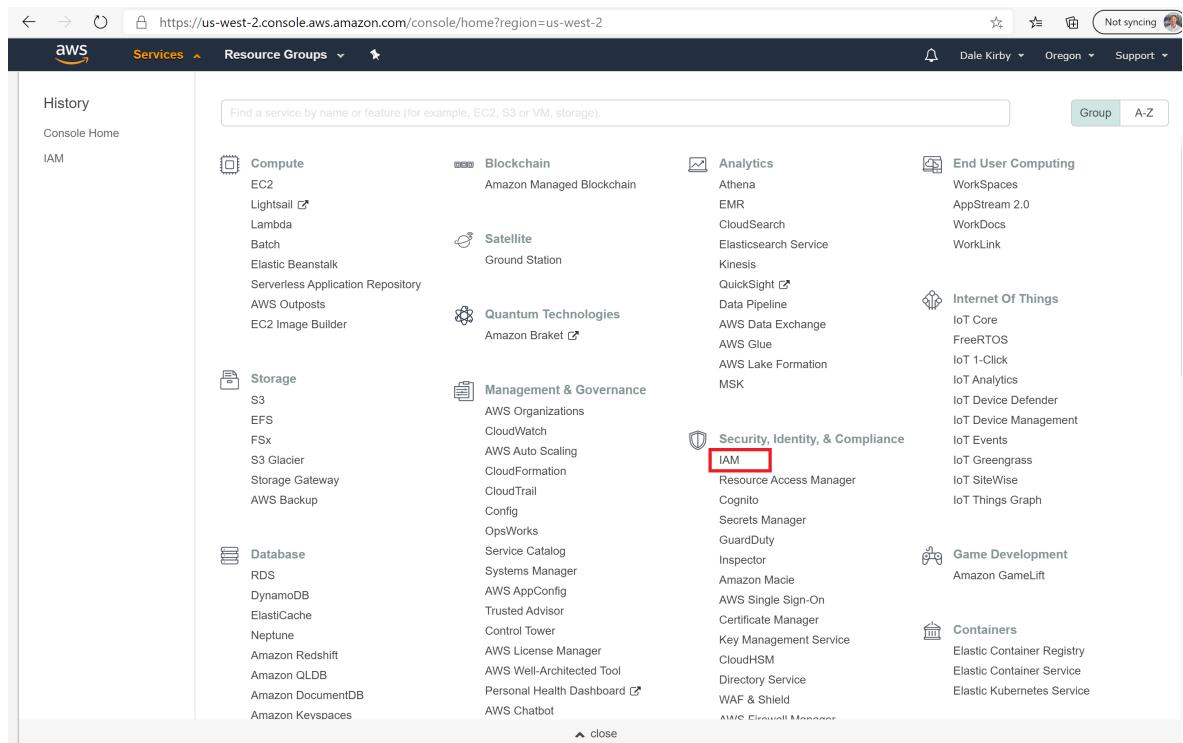
ⓘ Note

We highly recommend that you scope the service principal to a specific **Azure subscription and resource group**.

Create an AWS identity

In order for Terraform to create resources in AWS, we will need to create a new AWS IAM role with appropriate permissions and configure Terraform to use it.

1. Sign in to the [AWS management console](#)
2. After signing in, select the **Services** dropdown list in the top left. Under **Security, Identity, and Compliance**, select **IAM** to access the [identity and access management page](#)



The screenshot shows the AWS Identity and Access Management (IAM) console. The left sidebar includes links for Dashboard, Access management (Groups, Users, Roles, Policies, Identity providers, Account settings), Access reports (Access analyzer, Archive rules, Analyzers, Settings), Credential report, Organization activity, and Service control policies (SCPs). A search bar at the bottom of the sidebar contains the placeholder 'Search IAM'. The main content area is titled 'Welcome to Identity and Access Management'. It features a 'Sign-in link' field with a placeholder URL (https://[REDACTED]signin.aws.amazon.com/console) and a 'Customize' button. Below this is a 'IAM Resources' section showing 'Users: 0', 'Roles: 2', 'Groups: 0', and 'Identity Providers: 0'. A 'Customer Managed Policies' section is also present. The 'Security Status' section displays a progress bar indicating '0 out of 5 complete.' and a list of five items with warning icons: 'Delete your root access keys', 'Activate MFA on your root account', 'Create individual IAM users', 'Use groups to assign permissions', and 'Apply an IAM password policy'. An 'AWS CloudTrail Manager' link is visible at the bottom right of the security status area.

3. Click on **Users** from the left menu, and then select **Add user** to create a new IAM user.

Identity and Access Management (IAM)

Add user Delete user

Find users by username or access key

User name	Groups	Access key age

There are no IAM users. [Learn more](#)

4. On the **Add User** page, name the user `Terraform` and select the **Programmatic Access** checkbox, and then select **Next**.

Add user

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type* **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

AWS Management Console access
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

Cancel [Next: Permissions](#)

5. On the **Set Permissions** page, select **Attach existing policies directly** and then select the box next to **AmazonEC2FullAccess** as seen in the screenshot, and then select **Next**.

Add user

1 2 3 4 5

▼ Set permissions

Add user to group Copy permissions from existing user Attach existing policies directly

Create policy

	Policy name ▾	Type	Used as
<input type="checkbox"/>	▶ AmazonEC2ContainerServiceRole	AWS managed	None
<input checked="" type="checkbox"/>	▶ AmazonEC2FullAccess	AWS managed	Permissions policy (1)
<input type="checkbox"/>	▶ AmazonEC2ReadOnlyAccess	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RoleforAWSCodeDeploy	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RoleforDataPipelineRole	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RoleforSSM	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RolePolicyForLaunchWizard	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2SpotFleetAutoscaleRole	AWS managed	None

▶ Set permissions boundary

Cancel

6. On the **Tags** page, assign a tag with a key of `azure-arc-demo` and select **Next** to proceed to the **Review** page.

Add user

1 2 3 4 5

Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
azure-arc-demo		×
Add new key		

You can add 49 more tags.

[Cancel](#) [Previous](#) [Next: Review](#)

7. Verify that everything is correct and select **Create user** when ready.

Add user

1 2 3 4 5

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	terraform
AWS access type	Programmatic access - with an access key
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

Tags

The new user will receive the following tag

Key	Value
azure-arc-demo	(empty)

[Cancel](#) [Previous](#) [Create user](#)

8. After the user is created, you will see the user's access key ID and secret access key.

Copy these values before selecting **Close**. On the next page, you can see an example of what this should look like. Once you have these keys, you will be able to use them with Terraform to create AWS resources.

Add user

1 2 3 4 5

Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://976804645701.signin.aws.amazon.com/console>

 Download .csv

	User	Access key ID	Secret access key
▶	terraform	AKIA6G3QYBNCYHMF6JNN	SblzneQPfhlTpGOlvc6Hy9yvicGU6Mlh/8do7xm Hide

Configure Terraform

Before executing the Terraform plan, you must export the environment variables which will be used by the plan. These variables are based on your Azure subscription and tenant, the Azure service principal, and the AWS IAM user and keys you just created.

1. Retrieve your Azure subscription ID and tenant ID using the `az account list` command.
2. The Terraform plan creates resources in both Microsoft Azure and AWS. It then executes a script on an AWS EC2 virtual machine to install the Azure Arc agent and all necessary artifacts. This script requires certain information about your AWS and Azure environments. Edit `scripts/vars.sh` and update each of the variables with the appropriate values.

- `TF_VAR_subscription_id` = your Azure subscription ID
- `TF_VAR_client_id` = your Azure service principal application ID
- `TF_VAR_client_secret` = your Azure service principal password
- `TF_VAR_tenant_id` = your Azure tenant ID
- `AWS_ACCESS_KEY_ID` = AWS access key
- `AWS_SECRET_ACCESS_KEY` = AWS secret key

3. From the Azure CLI, navigate to the

`azure_arc_servers_jumpstart/aws/ubuntu/terraform` directory of the cloned repo.

4. Export the environment variables you edited by running `scripts/vars.sh` with the `source` command as shown below. Terraform requires these to be set for the plan to execute properly. Note that this script will also be automatically executed remotely on the AWS virtual machine as part of the Terraform deployment.

```
Console
```

```
source ./scripts/vars.sh
```

5. Make sure your SSH keys are available in `~/ssh` and named `id_rsa.pub` and `id_rsa`. If you followed the `ssh-keygen` guide above to create your key then this should already be set up correctly. If not, you may need to modify `main.tf` to use a key with a different path.

6. Run the `terraform init` command which will download the Terraform AzureRM provider.

```
!:/mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/ubuntu/terraform$ terraform init
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "local" (hashicorp/local) 1.4.0...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.7.0...
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 2.9.0...
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
!:/mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/ubuntu/terraform$
```

Deployment

1. Run the `terraform apply --auto-approve` command and wait for the plan to finish. Upon completion, you will have an AWS Amazon Linux 2 EC2 instance deployed and connected as a new Azure Arc-enabled server inside a new resource group.
2. Open the Azure portal and navigate to the `arc-aws-demo` resource group. The virtual machine created in AWS will be visible as a resource.

Semi-automated deployment (optional)

As you may have noticed, the last step of the run is to register the VM as a new Azure Arc-enabled server resource.

```

11 # Run connect command
12 azcmagent connect \
13   --service-principal-id ${TF_VAR_client_id} \
14   --service-principal-secret ${TF_VAR_client_secret} \
15   --tenant-id ${TF_VAR_tenant_id} \
16   --subscription-id ${TF_VAR_subscription_id} \
17   --location "${location}" \
18   --resource-group "${resourceGroup}" \

```

If you want to demo/control the actual registration process, do the following:

1. In the [install_arc_agent.sh.tpl](#) script template, comment out the `run connect` command section and save the file.

```

11  # Run connect command
12  # azcmagent connect \
13  #   --service-principal-id $TF_VAR_client_id \
14  #   --service-principal-secret $TF_VAR_client_secret \
15  #   --tenant-id $TF_VAR_tenant_id \
16  #   --subscription-id $TF_VAR_subscription_id \
17  #   --location "${location}" \
18  #   --resource-group "${resourceGroup}" \

```

2. Get the public IP of the AWS VM by running `terraform output`.

```

$ terraform output
ip = 34.83.189.233

```

3. SSH the VM using the `ssh ubuntu@xx.xx.xx.xx` where `xx.xx.xx.xx` is the host IP.

```

:/mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/ubuntu/terraform$ ssh ubuntu@34.220.29.213
The authenticity of host '34.220.29.213 (34.220.29.213)' can't be established.
ECDSA key fingerprint is SHA256:hgHanPJ1WF8+r0xFnUmdd+atzcfo60SH4hWwJ9iP2qc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '34.220.29.213' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1107-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

3 packages can be updated.
3 updates are security updates.

New release '18.04.4 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Thu May 28 15:48:31 2020 from 99.106.206.153
ubuntu@ip-10-0-35-157:~$ 

```

4. Export all the environment variables in [vars.sh](#).

```

ubuntu@ip-10-0-35-157:~$ export TF_VAR_subscription_id="XXXXXXXXXX"
ubuntu@ip-10-0-35-157:~$ export TF_VAR_client_id="XXXXXXXXXX"
ubuntu@ip-10-0-35-157:~$ export TF_VAR_client_secret="XXXXXXXXXX"
ubuntu@ip-10-0-35-157:~$ export TF_VAR_tenant_id="7XXXXXXXXXX"
ubuntu@ip-10-0-35-157:~$ export AWS_ACCESS_KEY_ID="/XXXXXXXXXX"
ubuntu@ip-10-0-35-157:~$ export AWS_SECRET_ACCESS_KEY="XXXXXXXXXX"
ubuntu@ip-10-0-35-157:~$ 

```

5. Run the following command:

Console

```

azcmagent connect --service-principal-id $TF_VAR_client_id --service-
principal-secret $TF_VAR_client_secret --resource-group "arc-aws-demo"
--tenant-id $TF_VAR_tenant_id --location "westus2" --subscription-id
$TF_VAR_subscription_id

```

```
ubuntu@ip-10-0-35-157:~$ azcmagent connect --service-principal-id $TF_VAR_client_id --service-principal-secret $TF_VAR_client_secret --resource-group "Arc-Servers-Demo" --tenant-id $TF_VAR_tenant_id --location "westus2" --subscription-id $TF_VAR_subscription_id
time="2020-05-28T15:53:31Z" level=info msg="Onboarding Machine. It usually takes a few minutes to complete. Sometimes it may take longer depending on network and server load status."
time="2020-05-28T15:54:18Z" level=info msg="Successfully Onboarded Resource to Azure" VM Id=a6f67d27-b311-49f5-97b9-b2a9e5bf1c12
ubuntu@ip-10-0-35-157:~$
```

- When complete, your VM will be registered with Azure Arc and visible in the resource group via the Azure portal.

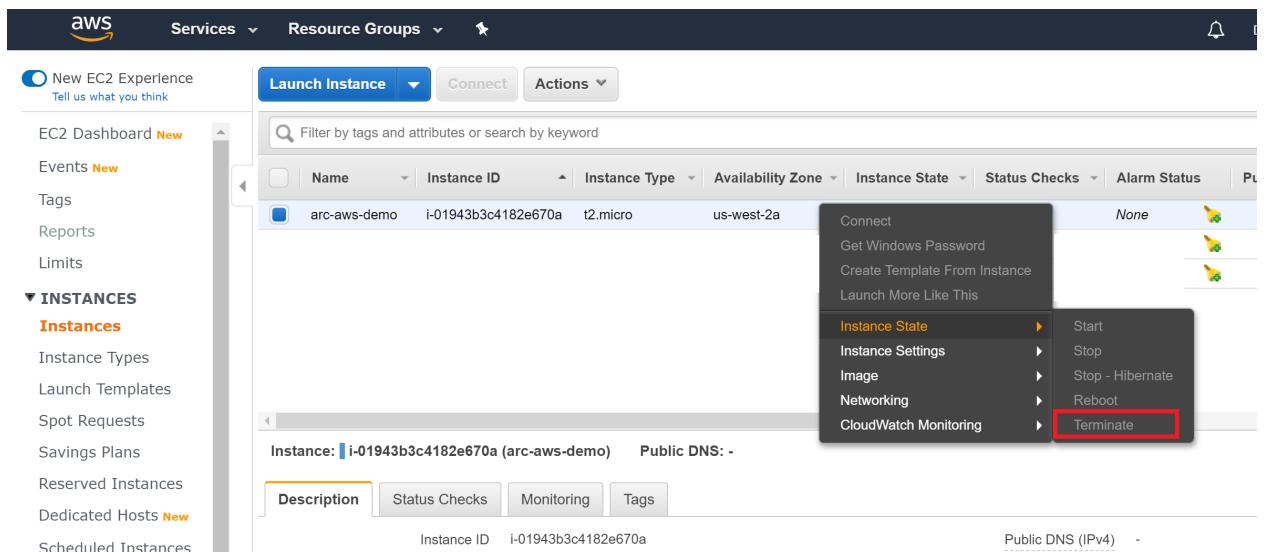
Delete the deployment

To delete all the resources you created as part of this demo use the `terraform destroy -auto-approve` command as shown below.

```
:/mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/ubuntu/terraform$ terraform destroy --auto-approve
local_file.install_arc_agent_sh: Refreshing state... [id=116d5901ecc6ab476ae485261b0f7da43dbc94d3]
aws_key_pair.keypair: Refreshing state... [id=terraform-2020052815470486900000001]
aws_vpc.vpc: Refreshing state... [id=vpc-0c413358bb3e5950d]
data.aws_ami.ubuntu: Refreshing state...
azurerm_resource_group.azure_rg: Refreshing state... [id=/subscriptions/2...)/resourceGroups/Arc-Servers-Demo
]
aws_internet_gateway.gw: Refreshing state... [id=igw-03130d0ac867ec82e]
aws_subnet.subnet1: Refreshing state... [id=subnet-0c995beb54b3a9661]
aws_security_group.ingress-all: Refreshing state... [id=sg-0503580d4ea9131b9e]
aws_instance.default: Refreshing state... [id=i-068f9b8dcacf3dd6b]
aws_default_route_table.route_table: Refreshing state... [id=rtb-0f547855fc1472a36]
local_file.install_arc_agent_sh: Destroying... [id=116d5901ecc6ab476ae485261b0f7da43dbc94d3]
local_file.install_arc_agent_sh: Destruction complete after 0s
aws_default_route_table.route_table: Destroying... [id=rtb-0f547855fc1472a36]
aws_default_route_table.route_table: Destruction complete after 0s
aws_instance.default: Destroying... [id=i-068f9b8dcacf3dd6b]
aws_internet_gateway.gw: Destroying... [id=igw-03130d0ac867ec82e]
azurerm_resource_group.azure_rg: Destroying... [id=/subscriptions/2...)/resourceGroups/Arc-Servers-Demo]
aws_instance.default: Still destroying... [id=i-068f9b8dcacf3dd6b, 10s elapsed]
aws_internet_gateway.gw: Still destroying... [id=igw-03130d0ac867ec82e, 10s elapsed]
azurerm_resource_group.azure_rg: Still destroying... [id=/subscriptions/2...)/resourceGroups/Arc-Servers-Demo, 1
0s elapsed]
azurerm_resource_group.azure_rg: Destruction complete after 1m50s

Destroy complete! Resources: 9 destroyed.
: /mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/ubuntu/terraform$
```

Alternatively, you can delete the AWS EC2 instance directly by terminating it from the [AWS console](#). Note that it will take a few minutes for the instance to actually be removed.



If you delete the instance manually, then you should also delete

`* ./scripts/install_arc_agent.sh`, which is created by the Terraform plan.

Use a Terraform plan to deploy an Amazon Linux 2 instance on Amazon Elastic Compute Cloud and connect it to Azure Arc

Article • 03/31/2023

This article provides guidance for using the provided [Terraform](#) plan to deploy an Amazon Web Services (AWS) Amazon Elastic Compute Cloud (EC2) Linux 2 instance and connect it as an Azure Arc-enabled server resource.

Prerequisites

1. Clone the Azure Arc Jumpstart repository.

Console

```
git clone https://github.com/microsoft/azure_arc.git
```

2. [Install or update Azure CLI](#). Azure CLI should be running version 2.7.0 or later. Use `az --version` to check your current installed version.

3. [Generate SSH key](#) (or use existing SSH key)

4. [Create free AWS account](#)

5. [Install Terraform >= 0.12](#)

6. Create an Azure service principal.

To connect the AWS virtual machine to Azure Arc, an Azure service principal assigned with the Contributor role is required. To create it, sign in to your Azure account and run the following commands:

Console

```
az login
az ad sp create-for-rbac -n "http://AzureArcAWS" --role contributor
```

Output should look similar to this:

JSON

```
{  
  "appId": "XXXXXXXXXXXXXXXXXXXXXX",  
  "displayName": "AzureArcAWS",  
  "name": "http://AzureArcAWS",  
  "password": "XXXXXXXXXXXXXXXXXXXXXX",  
  "tenant": "XXXXXXXXXXXXXXXXXXXXXX"  
}
```

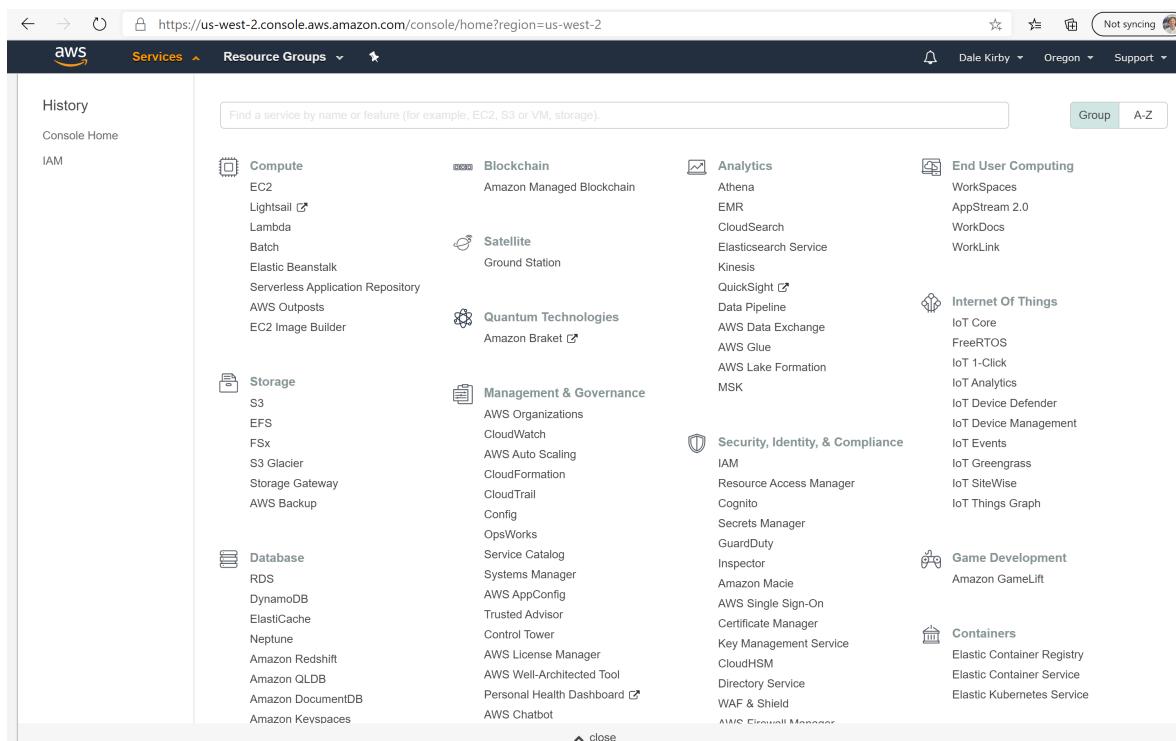
ⓘ Note

We highly recommend that you scope the service principal to a specific **Azure subscription and resource group**.

Create an AWS identity

In order for Terraform to create resources in AWS, we will need to create a new AWS IAM role with appropriate permissions and configure Terraform to use it.

1. Sign in to the [AWS management console](#)
2. After signing in, select the **Services** dropdown list in the top left. Under **Security, Identity, and Compliance**, select **IAM** to access the [identity and access management page](#)



The screenshot shows the AWS Identity and Access Management (IAM) dashboard. On the left, a sidebar lists various IAM management options like Groups, Users, Roles, Policies, and Identity providers. The 'Users' link under 'Access management' is highlighted. The main content area is titled 'Welcome to Identity and Access Management'. It displays basic resource counts: 'Users: 0', 'Groups: 0', 'Roles: 2', and 'Identity Providers: 0'. Below this is a 'Customer Managed Policies: 0' section. A 'Security Status' section contains five items with dropdown arrows: 'Delete your root access keys', 'Activate MFA on your root account', 'Create individual IAM users', 'Use groups to assign permissions', and 'Apply an IAM password policy'. At the bottom of the sidebar, there's a search bar labeled 'Search IAM' and an 'AWS account ID' field.

3. Click on **Users** from the left menu, and then select **Add user** to create a new IAM user.

The screenshot shows the 'Users' page within the IAM service. The left sidebar has 'Users' selected. The main area features a blue 'Add user' button and a red 'Delete user' button. Below them is a search bar with placeholder text 'Find users by username or access key'. A table header includes columns for 'User name' (with a dropdown arrow), 'Groups', and 'Access key age'. A message at the bottom states 'There are no IAM users. Learn more'.

4. On the **Add User** page, name the user `Terraform` and select the **Programmatic Access** checkbox, and then select **Next**

Add user

1 2 3 4 5

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type* **Programmatic access**

Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

AWS Management Console access

Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

[Cancel](#)

[Next: Permissions](#)

5. On the **Set Permissions** page, select **Attach existing policies directly** and then select the box next to **AmazonEC2FullAccess** as seen in the screenshot, and then select **Next**.

Add user

1 2 3 4 5

▼ Set permissions

Add user to group Copy permissions from existing user Attach existing policies directly

[Create policy](#) [Cancel](#)

	Policy name ▾	Type	Used as
<input type="checkbox"/>	▶ AmazonEC2ContainerServiceRole	AWS managed	None
<input checked="" type="checkbox"/>	▶ AmazonEC2FullAccess	AWS managed	Permissions policy (1)
<input type="checkbox"/>	▶ AmazonEC2ReadOnlyAccess	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RoleforAWSCodeDeploy	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RoleforDataPipelineRole	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RoleforSSM	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2RolePolicyForLaunchWizard	AWS managed	None
<input type="checkbox"/>	▶ AmazonEC2SpotFleetAutoscaleRole	AWS managed	None

▶ Set permissions boundary

[Cancel](#) [Previous](#) [Next: Tags](#)

6. On the **Tags** page, assign a tag with a key of `azure-arc-demo`, then select **Next** to proceed to the review page.

Add user

1 2 3 4 5

Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
azure-arc-demo		×
Add new key		

You can add 49 more tags.

[Cancel](#) [Previous](#) [Next: Review](#)

7. Verify that everything is correct, then select **Create user**.

Add user

1 2 3 4 5

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	terraform
AWS access type	Programmatic access - with an access key
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

Tags

The new user will receive the following tag

Key	Value
azure-arc-demo	(empty)

[Cancel](#) [Previous](#) [Create user](#)

8. After the user is created, you will see the user's access key ID and secret access key.

Copy these values down before selecting **Close**. On the next page, you can see an example of what this should look like. Once you have these keys, you will be able to use them with Terraform to create AWS resources.

Add user

1 2 3 4 5

 **Success**
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.
Users with AWS Management Console access can sign-in at: <https://976804645701.signin.aws.amazon.com/console>



	User	Access key ID	Secret access key
	terraform	AKIA6G3QYBNCYHMF6JNN	SblzneQPfhITpGOlvc6Hy9yvicGU6Mlh/8d07xm Hide

Configure Terraform

Before executing the Terraform plan, you must export the environment variables which will be used by the plan. These variables are based on your Azure subscription and tenant, the Azure service principal, and the AWS IAM user and keys you just created.

1. Retrieve your Azure subscription ID and tenant ID using the `az account list` command.
2. The Terraform plan creates resources in both Microsoft Azure and AWS. It then executes a script on an AWS EC2 virtual machine to install the Azure Arc agent and all necessary artifacts. This script requires certain information about your AWS and Azure environments. Edit `scripts/vars.sh` and update each of the variables with the appropriate values.

- `TF_VAR_subscription_id` = your Azure subscription ID
- `TF_VAR_client_id` = your Azure service principal application ID
- `TF_VAR_client_secret` = your Azure service principal password
- `TF_VAR_tenant_id` = your Azure tenant ID
- `AWS_ACCESS_KEY_ID` = AWS access key
- `AWS_SECRET_ACCESS_KEY` = AWS secret key

3. From the Azure CLI, navigate to the

`azure_arc_servers_jumpstart/aws/a12/terraform` directory of the cloned repo.

4. Export the environment variables you edited by running `scripts/vars.sh` with the `source` command as shown below. Terraform requires these to be set for the plan to execute properly. Note that this script will also be automatically executed remotely on the AWS virtual machine as part of the Terraform deployment.

```
Console
```

```
source ./scripts/vars.sh
```

5. Make sure your SSH keys are available in `~/.ssh` and named `id_rsa.pub` and `id_rsa`. If you followed the `ssh-keygen` guide above to create your key then this should already be set up correctly. If not, you may need to modify `main.tf` to use a key with a different path.

6. Run the `terraform init` command which will download the Terraform AzureRM provider.

```
:mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/a12/terraform$ terraform init
Initializing the backend...
Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "azurerm" (hashicorp/azurerm) 2.9.0...
- Downloading plugin for provider "local" (hashicorp/local) 1.4.0...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.7.0...
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
:mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/a12/terraform$
```

Deployment

1. Run the `terraform apply --auto-approve` command and wait for the plan to finish. Upon completion, you will have an AWS Amazon Linux 2 EC2 instance deployed and connected as a new Azure Arc-enabled server inside a new resource group.
2. Open the Azure portal and navigate to the `arc-servers-demo` resource group. The virtual machine created in AWS will be visible as a resource.

Arc-Servers-Demo

Resource group | Directory: Microsoft

Overview

Activity log

Access control (IAM)

Tags

Events

Subscription (change)
Microsoft Azure Internal Consumption

Subscription ID

Tags (change)
Click here to add tags

Filter by name... Type == all Location == all Add filter

Showing 1 to 1 of 1 records. Show hidden types

Name ↑	Type ↑↓	Location ↑↓
azure-arc-demo	Machine - Azure Arc	West US 2

No grouping

Semi-automated deployment (optional)

As you may have noticed, the last step of the run is to register the VM as a new Azure Arc-enabled server resource.

```
11 # Run connect command
12 azcmagent connect \
13   --service-principal-id ${TF_VAR_client_id} \
14   --service-principal-secret ${TF_VAR_client_secret} \
15   --tenant-id ${TF_VAR_tenant_id} \
16   --subscription-id ${TF_VAR_subscription_id} \
17   --location "${location}" \
18   --resource-group "${resourceGroup}" \
```

If you want to demo/control the actual registration process, do the following:

1. In the `install_arc_agent.sh.tpl` script template, comment out the `run connect command` section and save the file.

```
11 # Run connect command
12 # azcmagent connect \
13 #   --service-principal-id ${TF_VAR_client_id} \
14 #   --service-principal-secret ${TF_VAR_client_secret} \
15 #   --tenant-id ${TF_VAR_tenant_id} \
16 #   --subscription-id ${TF_VAR_subscription_id} \
17 #   --location "${location}" \
18 #   --resource-group "${resourceGroup}" \
```

2. Get the public IP of the AWS VM by running `terraform output`.

```
$ terraform output  
ip = 34.83.189.233
```

3. SSH to the VM using the `ssh ec2-user@xx.xx.xx.xx`, where `xx.xx.xx.xx` is the host IP.

```
[ec2-user@ip-10-0-44-218 ~]$ ssh ec2-user@34.214.18.103  
The authenticity of host '34.214.18.103 (34.214.18.103)' can't be established.  
ECDSA key fingerprint is SHA256:SeAtRiR4/T24KQrICTBolDWzrQcyHLC2f5dr4jJ3QkI.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '34.214.18.103' (ECDSA) to the list of known hosts.  
Last login: Wed May 27 19:30:21 2020 from 99-106-206-153.lightspeed.nwirla.sbcglobal.net  
  
[ec2-user@ip-10-0-44-218 ~]$  
  
https://aws.amazon.com/amazon-linux-2/  
1 package(s) needed for security, out of 16 available  
Run "sudo yum update" to apply all updates.  
[ec2-user@ip-10-0-44-218 ~]$
```

4. Export all the environment variables in `vars.sh`

```
[ec2-user@ip-10-0-44-218 tmp]$ export TF_VAR_subscription_id="  
[ec2-user@ip-10-0-44-218 tmp]$ export TF_VAR_client_id="  
[ec2-user@ip-10-0-44-218 tmp]$ export TF_VAR_client_secret="  
[ec2-user@ip-10-0-44-218 tmp]$ export TF_VAR_tenant_id="  
[ec2-user@ip-10-0-44-218 tmp]$ export AWS_ACCESS_KEY_ID="  
[ec2-user@ip-10-0-44-218 tmp]$ export AWS_SECRET_ACCESS_KEY="  
[ec2-user@ip-10-0-44-218 tmp]$
```

5. Run the following command:

Console

```
azcmagent connect --service-principal-id $TF_VAR_client_id --service-principal-secret $TF_VAR_client_secret --resource-group "Arc-Servers-Demo" --tenant-id $TF_VAR_tenant_id --location "westus2" --subscription-id $TF_VAR_subscription_id
```

```
[ec2-user@ip-10-0-44-218 tmp]$ azcmagent connect --service-principal-id $TF_VAR_client_id --service-principal-secret $TF_VAR_client_secret --resource-group "Arc-Servers-Demo" --tenant-id $TF_VAR_tenant_id --location "westus2" --subscription-id $TF_VAR_subscription_id  
time="2020-05-27T19:35:09Z" level=info msg="Onboarding Machine. It usually takes a few minutes to complete. Sometimes it may take longer depending on network and server load status."  
time="2020-05-27T19:35:29Z" level=info msg="Successfully Onboarded Resource to Azure" VM Id=46b33e2a-3ea7-4708-92ca-ef1184ffb6  
77  
[ec2-user@ip-10-0-44-218 tmp]$
```

6. When complete, your VM will be registered with Azure Arc and visible in the resource group via the Azure portal.

Delete the deployment

To delete all the resources you created as part of this demo, use the `terraform destroy --auto-approve` command as shown below.

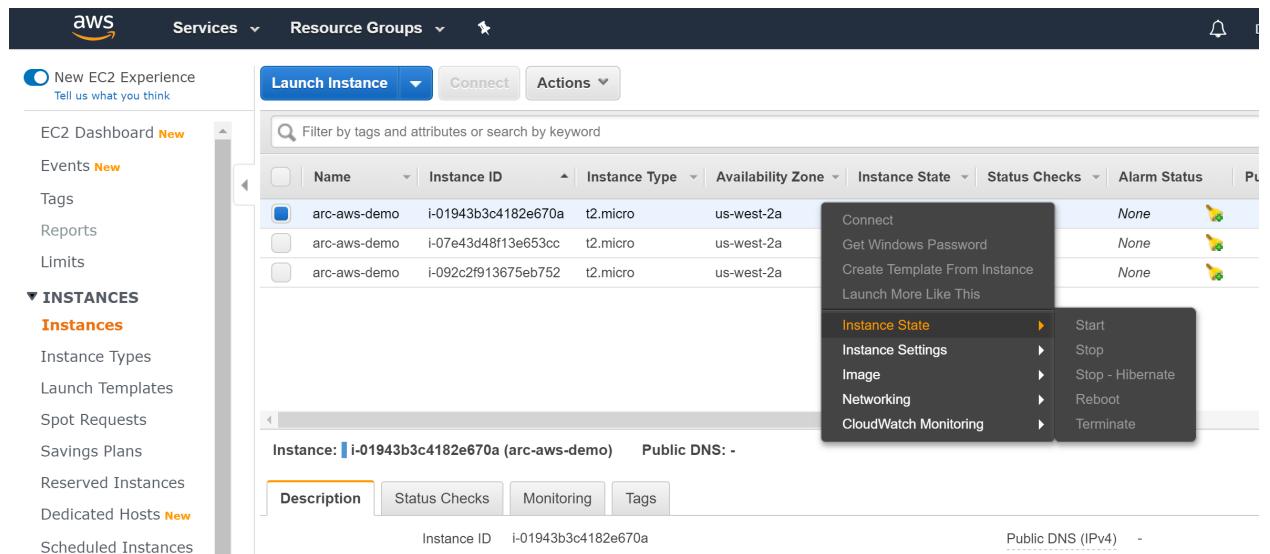
```

:/mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/a12/terraform$ terraform destroy --auto-approve
local_file.install_arc_agent_sh: Refreshing state... [id=d8cf0ade06f234186fdbb8c8ea8482289c5fd1e2]
aws_vpc.vpc: Refreshing state... [id=vpca-0a4aaaa2e7908a868]
aws_key_pair.keypair: Refreshing state... [id=terraform-20200527192930372600000001]
data.aws_ami.amazon-linux-2: Refreshing state...
azurerm_resource_group.azure_rg: Refreshing state... [id=/subscriptions/[REDACTED]/resourceGroups/Arc-Servers-Demo]
aws_subnet.subnet1: Refreshing state... [id=subnet-0263c391b5e12cc98]
aws_security_group.ingress-all: Refreshing state... [id=sg-024fd9a13138181e1]
aws_internet_gateway.gw: Refreshing state... [id=igw-0b438ccb31e6d86ac]
aws_default_route_table.route_table: Refreshing state... [id=rtb-084bf11929f384b02]
aws_instance.default: Refreshing state... [id=i-092c2f913675eb752]
local_file.install_arc_agent_sh: Destroying... [id=d8cf0ade06f234186fdbb8c8ea8482289c5fd1e2]
local_file.install_arc_agent_sh: Destruction complete after 0s
aws_instance.default: Destroying... [id=i-092c2f913675eb752]
aws_default_route_table.route_table: Destroying... [id=rtb-084bf11929f384b02]
aws_default_route_table.route_table: Destruction complete after 0s
aws_internet_gateway.gw: Destroying... [id=igw-0b438ccb31e6d86ac]
azurerm_resource_group.azure_rg: Destroying... [id=/subscriptions/[REDACTED]/resourceGroups/Arc-Servers-Demo]
aws_instance.default: Still destroying... [id=i-092c2f913675eb752, 10s elapsed]
aws_internet_gateway.gw: Still destroying... [id=igw-0b438ccb31e6d86ac, 10s elapsed]
azurerm_resource_group.azure_rg: Still destroying... [id=/subscriptions/[REDACTED]/resourceGroups/Arc-Servers-Demo, 10s elapsed]
aws_instance.default: Still destroying... [id=i-092c2f913675eb752, 20s elapsed]
aws_internet_gateway.gw: Still destroying... [id=igw-0b438ccb31e6d86ac, 20s elapsed]
azurerm_resource_group.azure_rg: Still destroying... [id=/subscriptions/[REDACTED]/resourceGroups/Arc-Servers-Demo, 20s elapsed]
aws_internet_gateway.gw: Destruction complete after 29s
aws_instance.default: Still destroying... [id=i-092c2f913675eb752, 30s elapsed]
aws_instance.default: Destruction complete after 31s
aws_key_pair.keypair: Destroying... [id=terraform-20200527192930372600000001]
aws_subnet.subnet1: Destroying... [id=subnet-0263c391b5e12cc98]
aws_security_group.ingress-all: Destroying... [id=sg-024fd9a13138181e1]
aws_key_pair.keypair: Destruction complete after 1s
aws_security_group.ingress-all: Destruction complete after 1s
aws_subnet.subnet1: Destruction complete after 1s
aws_vpc.vpc: Destroying... [id=vpca-0a4aaaa2e7908a868]
azurerm_resource_group.azure_rg: Still destroying... [id=/subscriptions/[REDACTED]/resourceGroups/Arc-Servers-Demo, 30s elapsed]
aws_vpc.vpc: Destruction complete after 1s
azurerm_resource_group.azure_rg: Destruction complete after 1m34s

Destroy complete! Resources: 9 destroyed.
:/mnt/c/dev/azure_arc/azure_arc_servers_jumpstart/aws/a12/terraform$ 

```

Alternatively, you can delete the AWS EC2 instance directly by terminating it from the [AWS console](#).



Use Ansible to scale onboarding Amazon Web Services Amazon Elastic Compute Cloud instances to Azure Arc

Article • 03/31/2023

This article provides guidance for using [Ansible](#) to scale onboarding Amazon Web Services (AWS) Amazon Elastic Compute Cloud (EC2) instances to Azure Arc.

This guide assumes that you have a basic understanding of Ansible. A basic Ansible playbook and configuration is provided that uses the [amazon.aws.aws_ec2](#) plug-in for dynamic loading of EC2 server inventory.

This guide can be used even if you do not already have an existing Ansible test environment and includes a Terraform plan that will create a sample AWS EC2 server inventory comprised of four Windows Server 2019 servers and four Ubuntu servers along with a basic CentOS 7 Ansible control server with a simple configuration.

⚠️ Warning

The provided Ansible sample workbook uses WinRM with password authentication and HTTP to configure Windows-based servers. This is not advisable for production environments. If you are planning to use Ansible with Windows hosts in a production environment then you should use **WinRM over HTTPS** with a certificate.

Prerequisites

1. Clone the Azure Arc Jumpstart repository.

Console

```
git clone https://github.com/microsoft/azure_arc.git
```

2. [Install or update Azure CLI to version 2.7 and above](#). Use the following command to check your current installed version.

Console

```
az --version
```

3. [Generate SSH key](#) (or use an existing SSH key)

4. [Create a free AWS account](#)

5. [Install Terraform >= v0.13](#)

- Create an Azure service principal.

To connect the AWS virtual machine to Azure Arc, an Azure service principal assigned with the Contributor role is required. To create it, sign in to your Azure account and run the following command. You can also run this command in [Azure Cloud Shell](#).

Console

```
az login  
az ad sp create-for-rbac -n "<Unique SP Name>" --role contributor
```

For example:

Console

```
az ad sp create-for-rbac -n "http://AzureArcAWS" --role  
contributor
```

Output should look like this:

JSON

```
{  
  "appId": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",  
  "displayName": "AzureArcAWS",  
  "name": "http://AzureArcAWS",  
  "password": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX",  
  "tenant": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"  
}
```

ⓘ Note

We highly recommend that you scope the service principal to a specific **Azure subscription and resource group**.

Create an AWS identity

In order for Terraform to create resources in AWS, we'll need to create a new AWS IAM role with appropriate permissions and configure Terraform to use it.

1. Sign in to the [AWS management console](#)

2. After signing in, select the **Services** dropdown list in the top left. Under **Security, Identity, and Compliance**, select **IAM** to access the [identity and access management page](#)

The screenshot shows the AWS Management Console navigation bar at the top with links for AWS, Services, Resource Groups, and a user profile. Below the bar is a search bar and a group of buttons labeled 'Group' and 'A-Z'. The main area is a grid of service icons and names. The 'Identity and Access Management (IAM)' service is highlighted with a red box. Other services visible include Compute, Blockchain, Analytics, End User Computing, Storage, Management & Governance, Security, Identity, & Compliance, Database, Game Development, and Containers.

The screenshot shows the IAM dashboard with a sidebar on the left containing links for Dashboard, Access management, Access reports, and other account settings. The main content area displays a welcome message, IAM users sign-in link, IAM Resources (Users: 0, Groups: 0, Roles: 2, Identity Providers: 0), Customer Managed Policies: 0, and a Security Status section with five items: Delete your root access keys, Activate MFA on your root account, Create individual IAM users, Use groups to assign permissions, and Apply an IAM password policy. At the bottom, there is a search bar and an AWS account ID field.

3. Select **Users** from the left menu, and then select **Add user** to create a new IAM user.

The screenshot shows the AWS Identity and Access Management (IAM) service. On the left, there's a navigation pane with several sections: Dashboard, Access management (Groups, Users, Roles, Policies, Identity providers, Account settings), Access reports (Access analyzer, Archive rules, Analyzers, Settings), and Credential report. The 'Users' section is currently selected. In the main content area, there are two buttons: 'Add user' (blue) and 'Delete user' (red). Below them is a search bar with the placeholder 'Find users by username or access key'. A table header is visible with columns for 'User name', 'Groups', and 'Access key age'. A message at the bottom states 'There are no IAM users. [Learn more](#)'.

4. On the **Add User** page, name the user **Terraform** and select the **Programmatic Access** checkbox, and then select **Next**.

The screenshot shows the 'Add user' process. Step 1, 'Set user details', is highlighted with a blue circle. It shows a 'User name*' field containing 'terraform'. Below it is a button '+ Add another user'. Step 2 through 5 are shown as smaller circles.

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

+ Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type* **Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

AWS Management Console access
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

Cancel **Next: Permissions**

5. On the next page, **Set Permissions**, select **Attach existing policies directly** then select the box next to **AmazonEC2FullAccess** as shown in the screenshot, and then select **Next**.

Add user

1 2 3 4 5

▼ Set permissions

Add user to group Copy permissions from existing user Attach existing policies directly

Create policy

Filter policies ▾ Search Showing 540 results

	Policy name ▾	Type	Used as
<input type="checkbox"/>	▶ AmazonEC2ContainerServiceRole	AWS managed	<i>None</i>
<input checked="" type="checkbox"/>	▶ AmazonEC2FullAccess	AWS managed	Permissions policy (1)
<input type="checkbox"/>	▶ AmazonEC2ReadOnlyAccess	AWS managed	<i>None</i>
<input type="checkbox"/>	▶ AmazonEC2RoleforAWSCodeDeploy	AWS managed	<i>None</i>
<input type="checkbox"/>	▶ AmazonEC2RoleforDataPipelineRole	AWS managed	<i>None</i>
<input type="checkbox"/>	▶ AmazonEC2RoleforSSM	AWS managed	<i>None</i>
<input type="checkbox"/>	▶ AmazonEC2RolePolicyForLaunchWizard	AWS managed	<i>None</i>
<input type="checkbox"/>	▶ AmazonEC2SpotFleetAutoscaleRole	AWS managed	<i>None</i>

▶ Set permissions boundary

 Cancel Previous Next: Tags

6. On the **Tags** page, assign a tag with a key of `azure-arc-demo` and select **Next** to proceed to the **Review** page.

Add user

1 2 3 4 5

Add tags (optional)

IAM tags are key-value pairs you can add to your user. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this user. [Learn more](#)

Key	Value (optional)	Remove
azure-arc-demo		×
Add new key		

You can add 49 more tags.

[Cancel](#) [Previous](#) [Next: Review](#)

7. Verify that everything is correct and select **Create user**.

Add user

1 2 3 4 5

Review

Review your choices. After you create the user, you can view and download the autogenerated password and access key.

User details

User name	terraform
AWS access type	Programmatic access - with an access key
Permissions boundary	Permissions boundary is not set

Permissions summary

The following policies will be attached to the user shown above.

Type	Name
Managed policy	AdministratorAccess

Tags

The new user will receive the following tag

Key	Value
azure-arc-demo	(empty)

[Cancel](#) [Previous](#) [Create user](#)

8. After the user is created, you will see the user's access key ID and secret access key.

Copy these values down before selecting **Close**. On the next page, you can see an example of what this should look like. Once you have these keys, you will be able to use them with Terraform to create AWS resources.

Add user

1 2 3 4 5

Success

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://976804645701.signin.aws.amazon.com/console>

 Download .csv

	User	Access key ID	Secret access key
▶	terraform	AKIA6G3QYBNCYHMF6JNN	SblzneQPfhlTpGOlvc6Hy9yvicGU6Mlh/8d07xm Hide

Option 1: Create a sample AWS server inventory and Ansible control server using Terraform and onboard the servers to Azure Arc

ⓘ Note

If you already have an existing AWS server inventory and Ansible server, skip to option 2.

Configure Terraform

Before executing the Terraform plan, you must export the environment variables which will be used by the plan. These variables are based on your Azure subscription and tenant, the Azure service principal, and the AWS IAM user and keys you just created.

1. Retrieve your Azure subscription ID and tenant ID using the `az account list` command.
2. The Terraform plan creates resources in both Microsoft Azure and AWS. It then executes a script on an AWS EC2 virtual machine to install Ansible and all necessary artifacts. This Terraform plan requires certain information about your AWS and

Azure environments which it accesses using environment variables. Edit [scripts/vars.sh](#) and update each of the variables with the appropriate values.

- `TF_VAR_subscription_id` = your Azure subscription ID
- `TF_VAR_client_id` = your Azure service principal application ID
- `TF_VAR_client_secret` = your Azure service principal password
- `TF_VAR_tenant_id` = your Azure tenant ID
- `AWS_ACCESS_KEY_ID` = AWS access key
- `AWS_SECRET_ACCESS_KEY` = AWS secret key

3. From your shell, navigate to the

`azure_arc_servers_jumpstart/aws/scaled_deployment/ansible/terraform`) directory of the cloned repository.

4. Export the environment variables you edited by running [scripts/vars.sh](#) with the source command as shown below. Terraform requires these to be set for the plan to execute properly.

Console

```
source ./scripts/vars.sh
```

5. Make sure your SSH keys are available in `~/.ssh` and named `id_rsa.pub` and `id_rsa`. If you followed the SSH keygen guide above to create your key then this should already be set up correctly. If not, you may need to modify [aws_infra.tf](#) to use a key with a different path.

6. Run the `terraform init` command which will download the required Terraform providers.

```
dakir@DESKTOP-SIUER8N:~/dev/azure_arc/azure_arc_servers_jumpstart/aws/scale_deployment/ansible/terraform$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding hashicorp/local versions matching "~> 1.4"...
- Finding hashicorp/http versions matching "~> 1.2.0"...
- Finding hashicorp/azurerm versions matching "~> 2.9.0"...
- Finding hashicorp/template versions matching "~> 2.1.2"...
- Finding hashicorp/aws versions matching "~> 2.7.0"...
- Installing hashicorp/aws v2.7.0...
- Installed hashicorp/aws v2.7.0 (signed by HashiCorp)
- Installing hashicorp/local v1.4.0...
- Installed hashicorp/local v1.4.0 (signed by HashiCorp)
- Installing hashicorp/http v1.2.0...
- Installed hashicorp/http v1.2.0 (signed by HashiCorp)
- Installing hashicorp/azurerm v2.9.0...
- Installed hashicorp/azurerm v2.9.0 (signed by HashiCorp)
- Installing hashicorp/template v2.1.2...
- Installed hashicorp/template v2.1.2 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
dakir@DESKTOP-SIUER8N:~/dev/azure_arc/azure_arc_servers_jumpstart/aws/scale_deployment/ansible/terraform$
```

Deploy server infrastructure

1. From the `azure_arc_servers_jumpstart/aws/scaled_deployment/ansible/terraform` directory, run `terraform apply --auto-approve` and wait for the plan to finish. Upon successful completion, you will have four Windows Server 2019 servers, four Ubuntu servers, and one CentOS 7 Ansible control server.
2. Open the AWS console and verify that you can see the created servers.

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm Status
aws-ansible-server	i-03af26407aaabf5f	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-ubuntu-1	i-000d85963160eba...	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-ubuntu-2	i-05844b4278c26e70	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-ubuntu-3	i-0bb46b8a9beed53...	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-ubuntu-4	i-0acc9e7787e6b7ff	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-windows-1	i-07bb3b78fb34859	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-windows-2	i-053ac3ed30eb791a3	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-windows-3	i-069fd3b31498f9e4c	t2.micro	us-west-2a	running	2/2 checks passed	None
aws-server-windows-4	i-03b68780eccb022a	t2.micro	us-west-2a	running	2/2 checks passed	None

Run the Ansible playbook to onboard the AWS EC2 instances as Azure Arc-enabled servers

- When the Terraform plan completes, it displays the public IP of the Ansible control server in an output variable named `ansible_ip`. SSH into the Ansible server by running `ssh centos@xx.xx.xx.xx`, where `xx.xx.xx.xx` is substituted for your Ansible server's IP address.

```
Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:

ansible_ip = 34.214.186.174
dakir@DESKTOP-SIUE8N:~/dev/azure_arc/azure_arc_servers_jumpstart/aws/scale_deployment/ansible/terraform$ ssh centos@34.214.186.174
The authenticity of host '34.214.186.174 (34.214.186.174)' can't be established.
ECDSA key fingerprint is SHA256:2/KqdyVMAUiQY7g10v28lCNAXie7Pdpxj0QzNdeBgw.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '34.214.186.174' (ECDSA) to the list of known hosts.
Last login: Sat Oct  3 21:23:24 2020 from 99-106-206-153.lightspeed.nwrlora.sbcglobal.net
[centos@ip-10-0-57-69 ~]$
```

- Change directory to the `ansible` directory by running `cd ansible`. This folder contains the sample Ansible configuration and the playbook we will use to onboard the servers to Azure Arc.

```
[centos@ip-10-0-57-69 ~]$ cd ansible
[centos@ip-10-0-57-69 ansible]$ ll
total 8
-rw-r--r--. 1 centos centos 174 Oct  3 21:23 ansible.cfg
drwxr-xr-x. 2 centos centos  43 Oct  3 21:23 ansible_plugins
-rw-r--r--. 1 centos centos 495 Oct  3 21:23 arc_agent.yml
drwxr-xr-x. 2 centos centos  83 Oct  3 21:23 group_vars
drwxr-xr-x. 2 centos centos  50 Oct  3 21:23 tasks
[centos@ip-10-0-57-69 ansible]$
```

- The `aw-ec2` Ansible plug-in requires AWS credentials to dynamically read your AWS server inventory. We will export these as environment variables. Run the following commands, replacing the values for `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` with the AWS credentials you created earlier.

Console

```
export AWS_ACCESS_KEY_ID="XXXXXXXXXXXXXXXXXXXX"
export AWS_SECRET_ACCESS_KEY="XXXXXXXXXXXXXXXXXXXX"
```

- Replace the placeholder values for Azure tenant ID and subscription ID in the `group-vars/all.yml` file with the appropriate values for your environment.

```
azure_arc_servers_jumpstart > aws > scale_deployment > ansible > terraform > ansible_config > group_vars > ! all.yml > ...
1   ---
2   |   # Variables here are applicable to all host groups
3   |
4   |   resource_group: "Arc-AWS-Demo"
5   |   tenant_id: "<Azure Tenant ID>"
6   |   azure_region: "westus2"
7   |   subscription_id: "<Azure Subscription ID>"
```

5. Run the Ansible playbook by executing the following command, substituting your Azure service principal ID and service principal secret.

```
Console

ansible-playbook arc_agent.yml -i ansible_plugins/inventory-uswest2-aws_ec2.yml --extra-vars '{"service_principal_id": "XXXXXXXX-XXXX-XXXXXX", "service_principal_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"}'
```

If the playbook run is successful, you should see output similar to the following screenshot.

```
PLAY RECAP ****
ec2-34-211-118-141.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-34-216-175-44.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-34-219-203-206.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-34-221-10-4.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-52-41-34-160.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-52-88-91-4.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-54-202-19-79.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-54-214-214-78.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0

[centos@ip-10-0-51-240 ansible]$
```

6. Open the Azure portal and navigate to the `arc-aws-demo` resource group. You should see the Azure Arc-enabled servers listed.

Name	Type	Location
aws-server-ubuntu-1	Server - Azure Arc	West US 2
aws-server-ubuntu-2	Server - Azure Arc	West US 2
aws-server-ubuntu-3	Server - Azure Arc	West US 2
aws-server-ubuntu-4	Server - Azure Arc	West US 2
aws-server-winsrv-1	Server - Azure Arc	West US 2
aws-server-winsrv-2	Server - Azure Arc	West US 2
aws-server-winsrv-3	Server - Azure Arc	West US 2
aws-server-winsrv-4	Server - Azure Arc	West US 2

Clean up environment by deleting resources

To delete all the resources you created as part of this demo, use the `terraform destroy --auto-approve` command as shown.

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
aws_instance.ubuntu[2]: Still destroying... [id=i-0f51d20a9d7825fd5, 2m20s elapsed]
aws_internet_gateway.gw: Still destroying... [id=igw-054544f55a9bbe6bd, 2m20s elapsed]
aws_instance.ubuntu[0]: Destruction complete after 2m27s
aws_instance.ubuntu[2]: Destruction complete after 2m27s
aws_instance.ubuntu[1]: Still destroying... [id=i-0166be9fe306e1f8f, 2m30s elapsed]
aws_instance.ubuntu[3]: Still destroying... [id=i-083add202d358c76d, 2m30s elapsed]
aws_internet_gateway.gw: Destruction complete after 2m29s
aws_instance.ubuntu[1]: Destruction complete after 2m37s
aws_instance.ubuntu[3]: Destruction complete after 2m38s
aws_key_pair.keypair: Destroying... [id=terraform-20201003235252845400000001]
aws_subnet.subnet1: Destroying... [id=subnet-001d8e838780eb476]
aws_security_group.ingress-all: Destroying... [id=sg-033388802fb36f6e5]
aws_key_pair.keypair: Destruction complete after 1s
aws_security_group.ingress-all: Destruction complete after 1s
aws_subnet.subnet1: Destruction complete after 1s
aws_vpc.vpc: Destroying... [id=vpc-0c1d3e9f60098a58e]
aws_vpc.vpc: Destruction complete after 1s
```

```
Destroy complete! Resources: 16 destroyed.
```

```
dakir@DESKTOP-SIUE8N:~/dev/azure_arc/azure_arc_servers_jumpstart/aws/scale_deployment/ansible/terraform$
```

Option 2: Onboarding an existing AWS server inventory to Azure Arc using your own Ansible control server

ⓘ Note

If you do not have an existing AWS server inventory and Ansible server, navigate back to option 1.

Review provided Ansible configuration and playbook

1. Navigate to the `ansible_config` directory and review the provided configuration.

The provided configuration contains a basic `ansible.cfg` file. This file enables the `amazon.aws.aws_ec2` Ansible plug-in which dynamically loads your server inventory by using an AWS IAM role. Ensure that the IAM role you are using has sufficient privileges to access the inventory you wish to onboard.

```
azure_arc_servers_jumpstart > aws > scale_deployment > ansible > terraform > ansible_config > ansible.cfg
1   [defaults]
2     inventory = ./ansible_plugins
3     enable_plugins = aws_ec2
4     host_key_checking = False
5     pipelining = True
6     log_path = ~/ansible/log/log
7     roles_path = ./roles
8     forks = 1000
9   |
```

2. The file `inventory-uswest2-aws_ec2.yml` configures the `aws_ec2` plug-in to pull inventory from `uswest2` region and group assets by applied tags. Adjust this file as

needed to support onboarding your server inventory, such as changing the region or adjusting groups or filters.

The files in `./ansible-config/group-vars` should be adjusted to provide the credentials you wish to use to onboard various Ansible host groups.

- When you have adjusted the provided configuration to support your environment, run the Ansible playbook by executing the following command, substituting your Azure service principal ID and service principal secret.

```
Console

ansible-playbook arc_agent.yml -i ansible_plugins/inventory-uswest2-
aws_ec2.yml --extra-vars '{"service_principal_id": "XXXXXXXX-XXXX-
XXXXXXXX", "service_principal_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"}'
```

As earlier, if the playbook run is successful, you should see an output that similar to the following screenshot:

```
PLAY RECAP ****
ec2-34-211-118-141.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-34-216-175-44.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-34-219-203-206.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-34-221-10-4.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-52-41-34-160.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-52-88-91-4.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-54-202-19-79.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0
ec2-54-214-214-78.us-west-2.compute.amazonaws.com : ok=5    changed=3    unreachable=0    failed=0    skipped=2    rescued=0    ignored=0

[centos@ip-10-0-51-240 ansible]$
```

As earlier, open Azure portal and navigate to the `arc-aws-demo` resource group. You should see the Azure Arc-enabled servers listed.

Name	Type	Location
aws-server-ubuntu-1	Server - Azure Arc	West US 2
aws-server-ubuntu-2	Server - Azure Arc	West US 2
aws-server-ubuntu-3	Server - Azure Arc	West US 2
aws-server-ubuntu-4	Server - Azure Arc	West US 2
aws-server-winsrv-1	Server - Azure Arc	West US 2
aws-server-winsrv-2	Server - Azure Arc	West US 2
aws-server-winsrv-3	Server - Azure Arc	West US 2
aws-server-winsrv-4	Server - Azure Arc	West US 2

Microsoft Entra identity management and access management for AWS

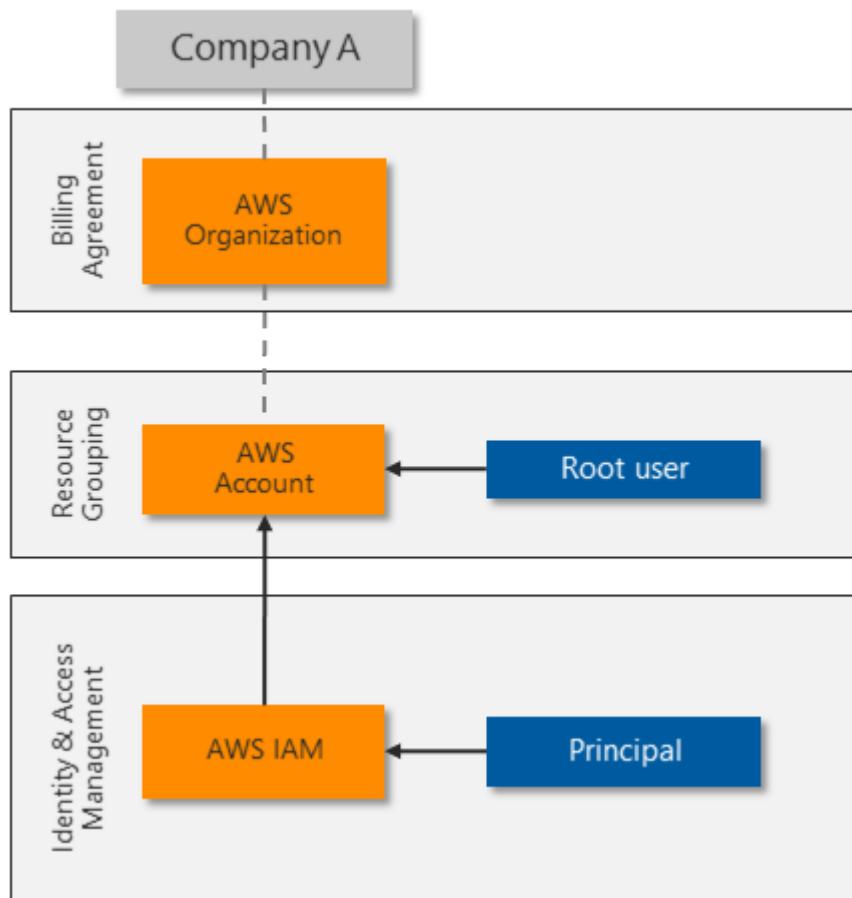
Azure

Microsoft Entra ID

This article provides AWS identity architects, administrators, and security analysts with immediate insights and detailed guidance for deploying Microsoft Entra identity and access solutions for AWS. You can configure and test these Microsoft security solutions without affecting your existing identity providers and AWS account users until you're ready to switch over.

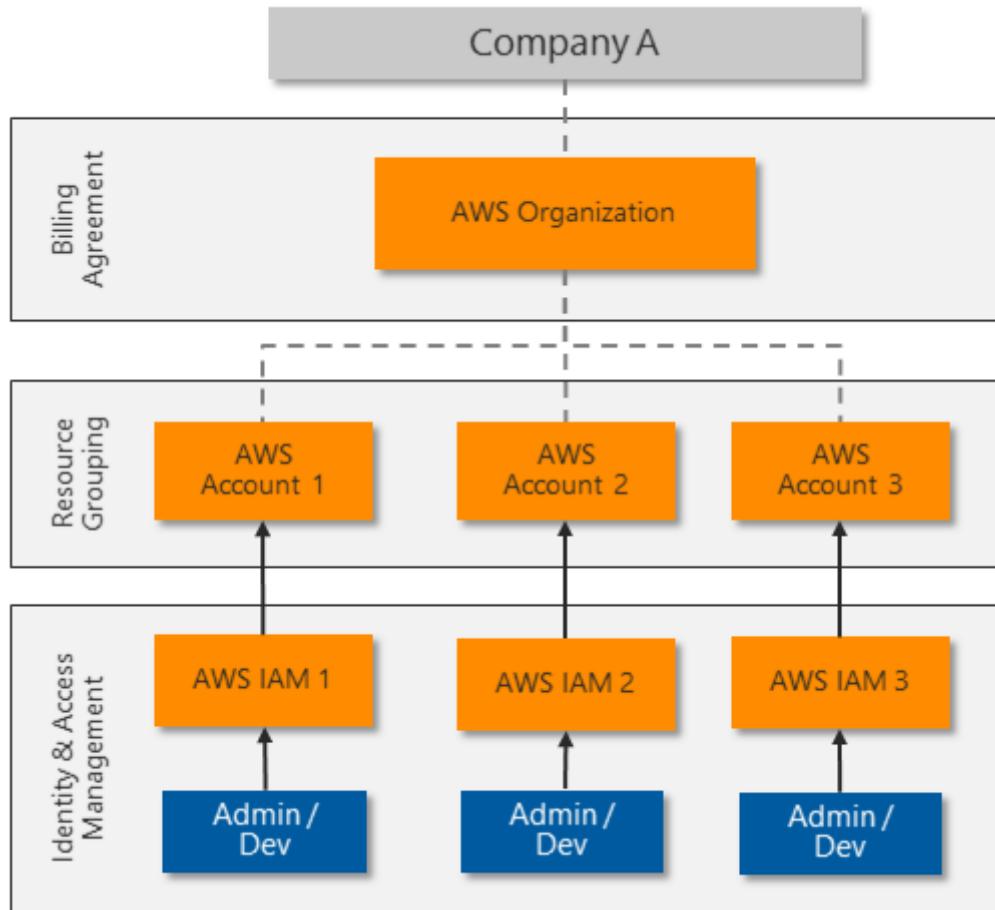
Architecture

AWS creates a separate *Identity and Access Management (IAM) store* for each account it creates. The following diagram shows the standard setup for an AWS environment with a single AWS account:



The *root user* fully controls the AWS account, and delegates access to other identities. The AWS IAM *principal* provides a unique identity for each role and user that needs to access the AWS account. AWS IAM can protect each root, principal, and user account with a complex password and basic MFA.

Many organizations need more than one AWS account, resulting in *identity silos* that are complex to manage:



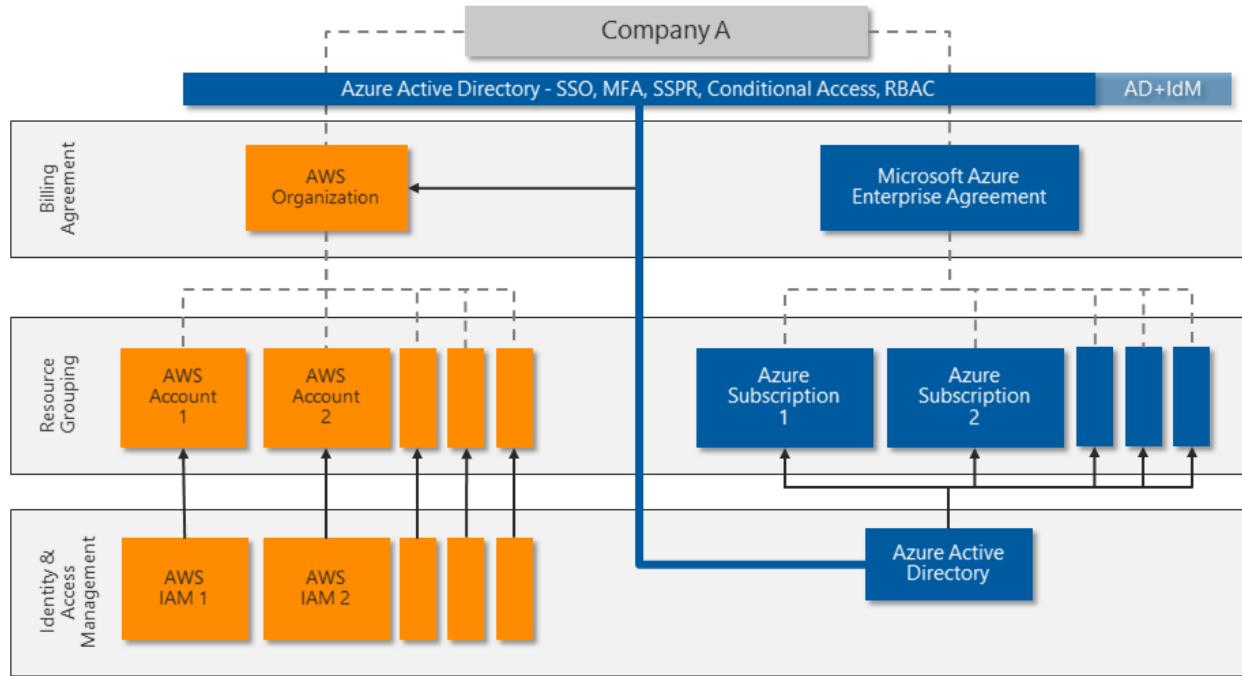
To allow centralized identity management and avoid having to manage multiple identities and passwords, most organizations want to use single sign-on for platform resources. Some AWS customers rely on server-based Microsoft Active Directory for SSO integration. Other customers invest in third-party solutions to synchronize or federate their identities and provide SSO.

Microsoft Entra ID provides centralized identity management with strong SSO authentication. Almost any app or platform that follows common web authentication standards, including AWS, can use Microsoft Entra ID for identity and access management.

Many organizations already use Microsoft Entra ID to assign and protect Microsoft 365 or hybrid cloud identities. Employees use their Microsoft Entra identities to access email, files, instant messaging, cloud applications, and on-premises resources. You can quickly

and easily integrate Microsoft Entra ID with your AWS accounts to let administrators and developers sign in to your AWS environments with their existing identities.

The following diagram shows how Microsoft Entra ID can integrate with multiple AWS accounts to provide centralized identity and access management:



Microsoft Entra ID offers several capabilities for direct integration with AWS:

- SSO across legacy, traditional, and modern authentication solutions.
- MFA, including integration with several third-party solutions from [Microsoft Intelligent Security Association \(MISA\)](#) partners.
- Powerful *Conditional Access* features for strong authentication and strict governance. Microsoft Entra ID uses Conditional Access policies and risk-based assessments to authenticate and authorize user access to the AWS Management Console and AWS resources.
- Large-scale threat detection and automated response. Microsoft Entra ID processes over 30 billion authentication requests per day, along with trillions of signals about threats worldwide.
- *Privileged Access Management (PAM)* to enable *Just-In-Time (JIT) provisioning* to specific resources.

Advanced Microsoft Entra identity management with AWS accounts

Other advanced Microsoft Entra features can provide extra layers of control for the most sensitive AWS accounts. Microsoft Entra ID P2 licenses include these advanced features:

- **Privileged Identity Management (PIM)** to provide advanced controls for all delegated roles within Azure and Microsoft 365. For example, instead of an administrator always using the Global Admin role, they have permission to activate the role on demand. This permission deactivates after a set time limit (one hour, for example). PIM logs all activations and has other controls that can further restrict the activation capabilities. PIM further protects your identity architecture by ensuring extra layers of governance and protection before administrators can make changes.

You can expand PIM to any delegated permission by controlling access to custom groups, such as the ones you created for access to AWS roles. For more information about deploying PIM, see [Deploy Microsoft Entra Privileged Identity Management](#).

- **Advanced Identity Protection** increases Microsoft Entra sign-in security by monitoring user or session risk. User risk defines the potential of the credentials being compromised, such as the user ID and password appearing in a publicly released breach list. Session risk determines whether the sign-in activity comes from a risky location, IP address, or other indicator of compromise. Both detection types draw on Microsoft's comprehensive threat intelligence capabilities.

For more information about Advanced Identity Protection, see the [Microsoft Entra ID Protection security overview](#).

- **Microsoft Defender for Identity** protects identities and services running on Active Directory domain controllers by monitoring all activity and threat signals. Defender for Identity identifies threats based on real-life experience from investigations of customer breaches. Defender for Identity monitors user behavior and recommends attack surface reductions to prevent advanced attacks like reconnaissance, lateral movement, and domain dominance.

For more information about Defender for Identity, see [What is Microsoft Defender for Identity](#).

Scenario details

Amazon Web Services (AWS) accounts that support critical workloads and highly sensitive information need strong identity protection and access control. AWS identity management is enhanced when combined with Microsoft Entra ID. Microsoft Entra ID is a cloud-based, comprehensive, centralized identity and access management solution that can help secure and protect AWS accounts and environments. Microsoft Entra ID provides centralized *single sign-on (SSO)* and strong authentication through *multi-factor*

authentication (MFA) and Conditional Access policies. Microsoft Entra ID supports AWS identity management, role-based identities, and access control.

Many organizations that use AWS already rely on Microsoft Entra ID for Microsoft 365 or hybrid cloud identity management and access protection. These organizations can quickly and easily use Microsoft Entra ID with their AWS accounts, often without extra cost. Other, [advanced Microsoft Entra features](#) like Privileged Identity Management (PIM) and Advanced Identity Protection can help protect the most sensitive AWS accounts.

Microsoft Entra ID easily integrates with other Microsoft security solutions, like Microsoft Defender for Cloud Apps and Microsoft Sentinel. For more information, see [Defender for Cloud Apps and Microsoft Sentinel for AWS](#). Microsoft security solutions are extensible and have multiple levels of protection. Organizations can implement one or more of these solutions along with other types of protection for a full security architecture that protects current and future AWS deployments.

Recommendations

Security

The following principles and guidelines are important for any cloud security solution:

- Ensure that the organization can monitor, detect, and automatically protect user and programmatic access into cloud environments.
- Continually review current accounts to ensure identity and permission governance and control.
- Follow [least privilege](#) and [zero trust](#) principles. Make sure that each user can access only the specific resources they require, from trusted devices and known locations. Reduce the permissions of every administrator and developer to provide only the rights they need for the role they're performing. Review regularly.
- Continuously monitor platform configuration changes, especially if they provide opportunities for privilege escalation or attack persistence.
- Prevent unauthorized data exfiltration by actively inspecting and controlling content.
- Take advantage of solutions you might already own like Microsoft Entra ID P2 that can increase security without more expense.

Basic AWS account security

To ensure basic security hygiene for AWS accounts and resources:

- Review the AWS security guidance at [Best practices for securing AWS accounts and resources](#).
- Reduce the risk of uploading and downloading malware and other malicious content by actively inspecting all data transfers through the AWS Management Console. Content that uploads or downloads directly to resources within the AWS platform, such as web servers or databases, might need more protection.
- Consider protecting access to other resources, including:
 - Resources created within the AWS account.
 - Specific workload platforms, like Windows Server, Linux Server, or containers.
 - Devices that administrators and developers use to access the AWS Management Console.

AWS IAM security

A key aspect of securing the AWS Management Console is controlling who can make sensitive configuration changes. The AWS account root user has unrestricted access. The security team should fully control the root user account to prevent it from signing in to the AWS Management Console or working with AWS resources.

To control the root user account:

- Consider changing the root user sign-in credentials from an individual's email address to a service account that the security team controls.
- Make sure the root user account password is complex, and enforce MFA for the root user.
- Monitor logs for instances of the root user account being used to sign in.
- Use the root user account only in emergencies.
- Use Microsoft Entra ID to implement delegated administrative access rather than using the root user for administrative tasks.

Clearly understand and review other AWS IAM account components for appropriate mapping and assignments.

- By default, an AWS account has no *IAM users* until the root user creates one or more identities to delegate access. A solution that synchronizes existing users from another identity system, such as Microsoft Active Directory, can also automatically provision IAM users.

- *IAM policies* provide delegated access rights to AWS account resources. AWS provides over 750 unique IAM policies, and customers can also define custom policies.
- *IAM roles* attach specific policies to identities. Roles are the way to administer *role-based access control (RBAC)*. The current solution uses [External Identities](#) to implement Microsoft Entra identities by assuming IAM roles.
- *IAM groups* are also a way to administer RBAC. Instead of assigning IAM policies directly to individual IAM users, create an IAM group, assign permissions by attaching one or more IAM policies, and add IAM users to the group to inherit the appropriate access rights to resources.

Some *IAM service accounts* must continue to run in AWS IAM to provide programmatic access. Be sure to review these accounts, securely store and restrict access to their security credentials, and rotate the credentials regularly.

Deploy this scenario

This next section shows you how to deploy Microsoft Entra ID for single sign-on to an individual AWS account.

Plan and prepare

To prepare for deployment of Azure security solutions, review and record current AWS account and Microsoft Entra information. If you've more than one AWS account deployed, repeat these steps for each account.

1. In the [AWS Billing Management Console](#), record the following current AWS account information:

- **AWS Account Id**, a unique identifier.
- **Account Name** or root user.
- **Payment method**, whether assigned to a credit card or a company billing agreement.
- **Alternate contacts** who have access to AWS account information.
- **Security questions** securely updated and recorded for emergency access.
- **AWS regions** enabled or disabled to comply with data security policy.

2. In the [AWS IAM Management Console](#), review and record the following AWS IAM components:

- **Groups** that have been created, including detailed membership and role-based mapping policies attached.
- **Users** that have been created, including the **Password age** for user accounts, and the **Access key age** for service accounts. Also confirm that MFA is enabled for each user.
- **Roles**. There are two default service-linked roles, **AWSServiceRoleForSupport** and **AWSServiceRoleForTrustedAdvisor**. Record any other roles, which are custom. These roles link to permission policies, to use for mapping roles in Microsoft Entra ID.
- **Policies**. Out-of-the-box policies have **AWS managed**, **Job function**, or **Customer managed** in the **Type** column. Record all other policies, which are custom. Also record where each policy is assigned, from the entries in the **Used as** column.
- **Identity providers**, to understand any existing Security Assertion Markup Language (SAML) identity providers. Plan how to replace the existing identity providers with the single Microsoft Entra identity provider.

3. In the [Azure portal](#), review the Microsoft Entra tenant:

- Assess **Tenant information** to see whether the tenant has a Microsoft Entra ID P1 or P2 license. A P2 license provides [Advanced Microsoft Entra identity management](#) features.
- Assess **Enterprise applications** to see whether any existing applications use the AWS application type, as shown by `http://aws.amazon.com/` in the **Homepage URL** column.

Plan Microsoft Entra deployment

The Microsoft Entra deployment procedures assume that Microsoft Entra ID is already configured for the organization, such as for a Microsoft 365 implementation. Accounts can be synchronized from an Active Directory domain, or can be cloud accounts created directly in Microsoft Entra ID.

Plan RBAC

If the AWS installation uses IAM groups and roles for RBAC, you can map the existing RBAC structure to new Microsoft Entra user accounts and security groups.

If the AWS account doesn't have a strong RBAC implementation, start by working on the most sensitive access:

1. Update the AWS account root user.

2. Review the AWS IAM users, groups, and roles that are attached to the IAM policy **AdministratorAccess**.
3. Work through the other assigned IAM policies, starting with policies that can modify, create, or delete resources and other configuration items. You can identify policies in use by looking at the **Used as** column.

Plan migration

Microsoft Entra ID centralizes all authentication and authorization. You can plan and configure user mapping and RBAC without affecting administrators and developers until you're ready to enforce the new methods.

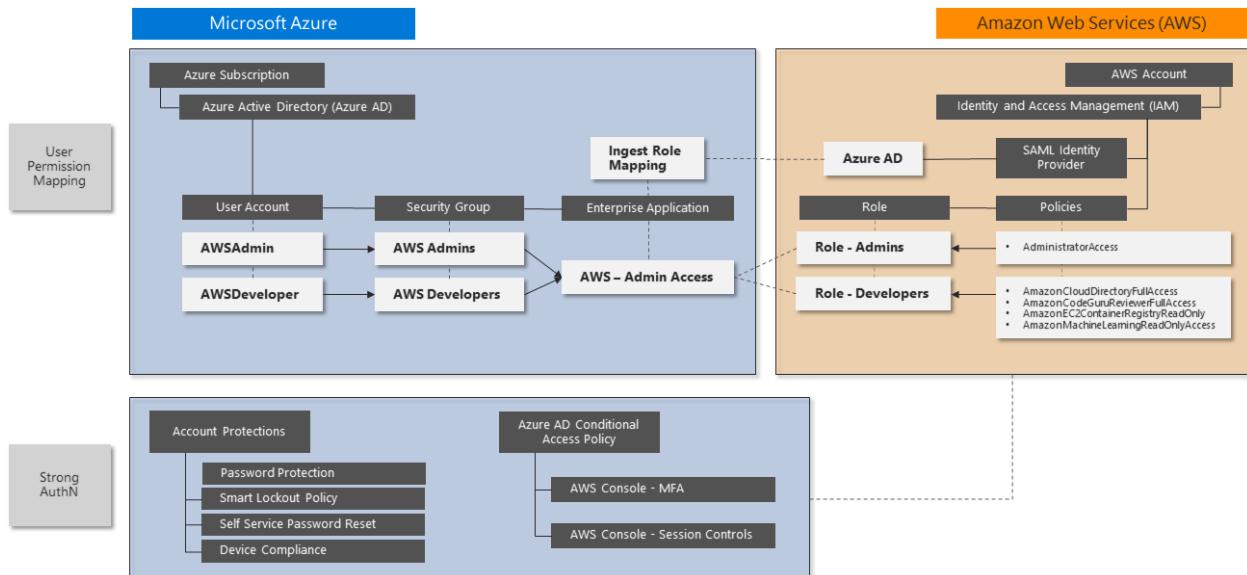
The high-level process for migrating from AWS IAM accounts to Microsoft Entra ID is as follows. For detailed instructions, see [Deployment](#).

1. Map IAM policies to Microsoft Entra roles, and use RBAC to map roles to security groups.
2. Replace each IAM user with a Microsoft Entra user who is a member of the appropriate security groups to sign in and gain appropriate permissions.
3. Test by asking each user to sign in to AWS with their Microsoft Entra account and confirm that they have the appropriate access level.
4. Once the user confirms Microsoft Entra ID access, remove the AWS IAM user account. Repeat the process for each user until they're all migrated.

For service accounts and programmatic access, use the same approach. Update each application that uses the account to use an equivalent Microsoft Entra user account instead.

Make sure any remaining AWS IAM users have complex passwords with MFA enabled, or an access key that's replaced regularly.

The following diagram shows an example of the configuration steps and final policy and role mapping across Microsoft Entra ID and AWS IAM:



Single sign-on integration

Microsoft Entra ID supports single sign-on integration with AWS SSO. You can connect Microsoft Entra ID to AWS in one place and centrally govern access across hundreds of accounts and AWS SSO integrated applications. This capability enables seamless Microsoft Entra sign-in experience for users to use the AWS CLI.

The following Microsoft security solution procedure implements SSO for the example roles **AWS Administrators** and **AWS Developers**. Repeat this process for any other roles you need.

This procedure covers the following steps:

1. Create a new Microsoft Entra enterprise application.
2. Configure Microsoft Entra SSO for AWS.
3. Update role mapping.
4. Test Microsoft Entra SSO into AWS Management Console.

The following links provide full detailed implementation steps and troubleshooting:

- [Microsoft tutorial: Microsoft Entra SSO integration with AWS](#)
- [AWS tutorial: Microsoft Entra ID to AWS SSO using the SCIM protocol ↗](#)

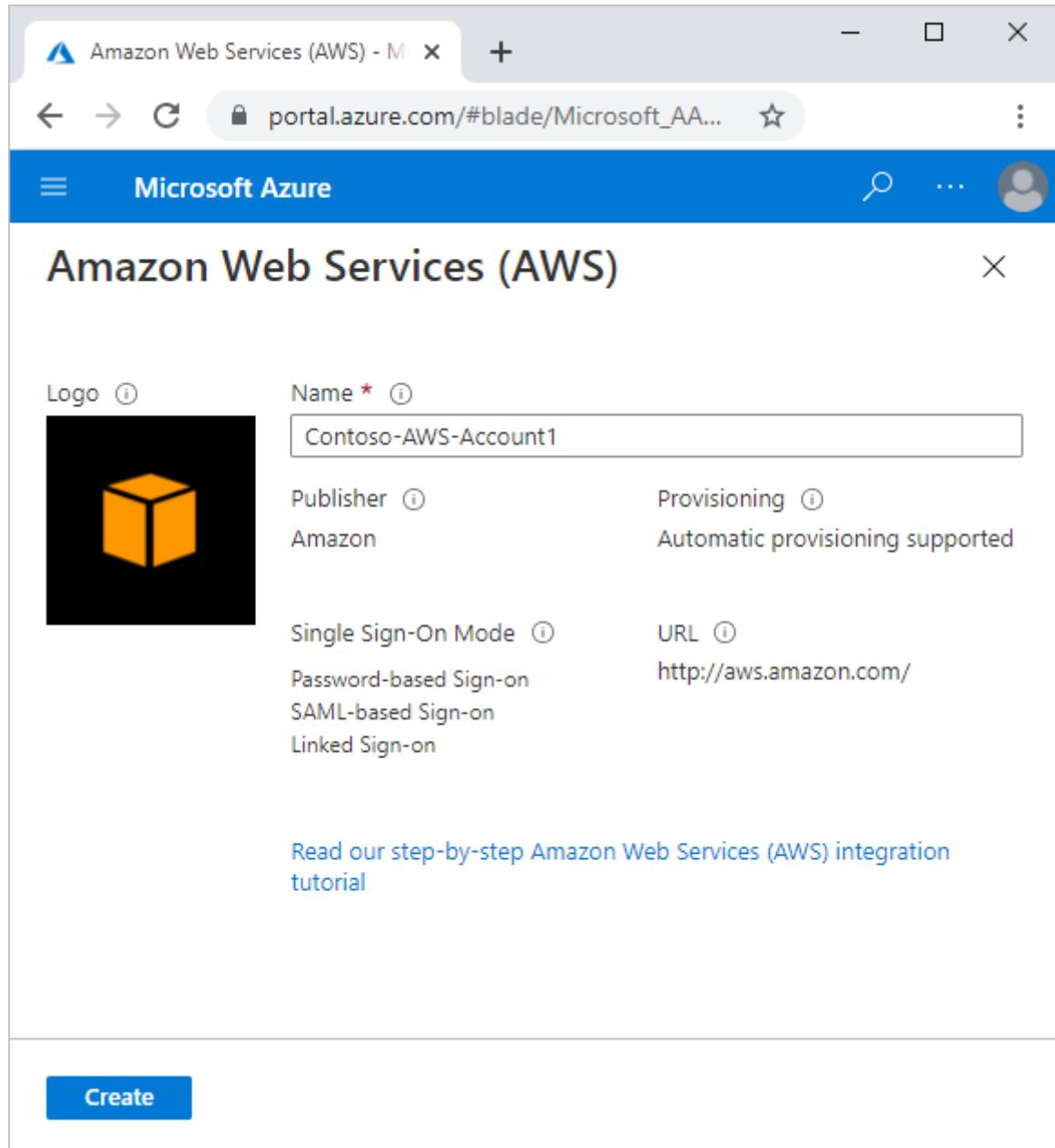
Add an AWS app to your Microsoft Entra enterprise applications

AWS administrators and developers use an enterprise application to sign in to Microsoft Entra ID for authentication, then redirect to AWS for authorization and access to AWS resources. The simplest method to see the application is by signing in to

<https://myapps.microsoft.com>, but you can also publish the unique URL anywhere that provides easy access.

Follow the instructions in [add Amazon Web Services \(AWS\) from the gallery](#) to set up the enterprise application. These instructions will let you know what AWS app to add to your Microsoft Entra enterprise applications.

If there's more than one AWS account to administer, such as DevTest and Production, use a unique name for the enterprise application that includes an identifier for the company and specific AWS account.



The screenshot shows the Microsoft Azure portal interface with the title "Amazon Web Services (AWS)". The configuration page for the AWS application is displayed, showing the following details:

- Logo**: An orange cube icon on a black background.
- Name ***: Contoso-AWS-Account1
- Publisher**: Amazon
- Provisioning**: Automatic provisioning supported
- Single Sign-On Mode**: Password-based Sign-on, SAML-based Sign-on, Linked Sign-on
- URL**: <http://aws.amazon.com/>

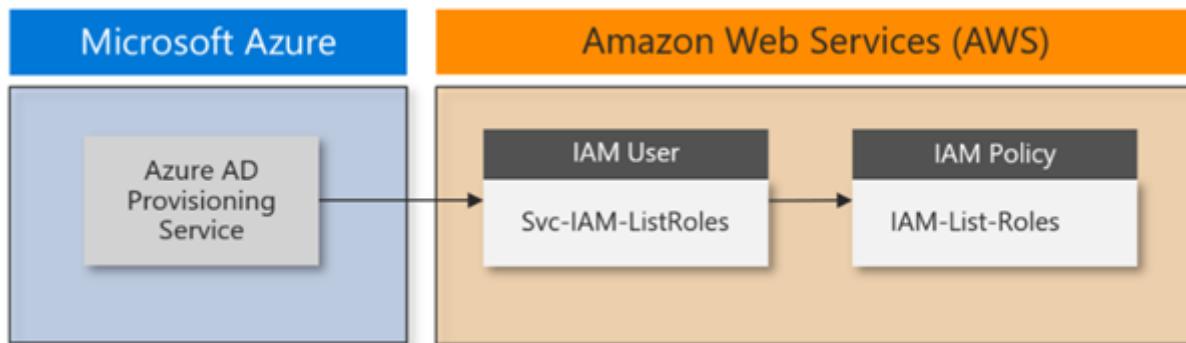
At the bottom left is a blue "Create" button.

Configure Microsoft Entra SSO for AWS

Follow the steps below to configure Microsoft Entra SSO for AWS:

1. On **Azure Portal**, follow the steps on [Configure Microsoft Entra SSO](#) to configure the **Enterprise Application** you've created for single sign-on to AWS.
2. On **AWS Console**, follow the steps on [Configure AWS SSO](#) to configure your AWS account for single sign-on. As part of this configuration, you'll create a new IAM user that acts on behalf of the Microsoft Entra provisioning agent to allow synchronization of all available **AWS IAM roles** into **Microsoft Entra ID**. AWS needs this IAM user to map users to roles before they can sign in to the **AWS Management Console**.
 - Make it easy to identify the components you create to support this integration. For example, name service accounts with a standard naming convention like "Svc-".
 - Be sure to document all new items.
 - Make sure any new credentials include complex passwords that you store centrally for secure lifecycle management.

Based on these configuration steps, you can diagram the interactions like this:



On **AWS Console**, follow the steps below to create more roles.

1. In **AWS IAM**, select **Roles** -> **Create Role**.
2. On the **Create role** page, perform the following steps:
 - a. Under **Select type of trusted entity**, select **SAML 2.0 federation**.
 - b. Under **Choose a SAML 2.0 Provider**, select the SAML provider you created in the previous step.
 - c. Select **Allow programmatic and AWS Management Console access**.
 - d. Select **Next: Permissions**.
3. On the **Attach permissions policies** dialog box, select **AdministratorAccess**. Then select **Next: Tags**.
4. In the **Add Tags** dialog box, leave it blank and select **Next: Review**.
5. In the **Review** dialog box, perform the following steps:
 - a. In **Role Name**, enter your role name (**Administrator**).

- b. In **Role Description**, enter the description.
- c. Select **Create Role**.
6. Create another role by following the steps listed above. Name the role **Developer** and give it a few selected permissions of your choice (such as **AmazonS3FullAccess**).
- You've successfully created an **Administrator** and a **Developer** role in AWS.
7. Create the following users and groups in **Microsoft Entra ID**:
- **User 1:** Test-AWSAdmin
 - **User 2:** Test-AWSDeveloper
 - **Group 1:** AWS-Account1-Administrators
 - **Group 2:** AWS-Account1-Developers
 - **Add** Test-AWSAdmin as a member of **AWS-Account1-Administrators**
 - **Add** Test-AWSDeveloper as a member of **AWS-Account1-Developers**
8. Follow the steps on [How to configure role provisioning in AWS Single-Account Access](#) to configure automated role provisioning. It can take up to one hour to complete the first provisioning cycle.

How to update role mapping

Because you're using two roles, perform these extra steps:

1. Confirm that the provisioning agent can see at least two roles:

The screenshot shows the Microsoft Entra ID Provisioning Overview page. The left sidebar has sections for Overview, Deployment Plan, and Diagnose and solve problems. Under Manage, the 'Provisioning' tab is selected. The main area shows the 'Current cycle status' as 'Incremental cycle completed.' with a progress bar at 100% complete. A red box highlights the 'Roles' section, which shows a count of 2. Below this, there's a 'View provisioning logs' link. On the right, under 'Statistics to date', there are links for 'View provisioning details' and 'View technical information'. At the top, there are buttons for Start provisioning, Stop provisioning, Restart provisioning, and Edit provisioning.

2. Go to **Users and groups** and select **Add User**.

3. Select **AWS-Account1-Administrators**.

4. Select the associated role.

⚠️ When you assign a group to an application, only users directly in the group will have access. The assignment does not cascade to nested groups.

Users and groups
1 group selected.

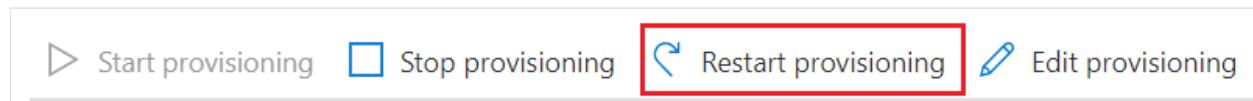
*Select a role
None Selected

AzureAD-AWSAdministrators,AzureAD-Developer
AzureAD-Developers,AzureAD-Developers

5. Repeat the preceding steps for each group-role mapping. Once complete, you should have two Microsoft Entra groups correctly mapped to AWS IAM roles:

Display Name	Object Type	Role assigned
<input type="checkbox"/> AW AWS-Account1-Administrators	Group	AzureAD-AWSAdministrators,AzureAD-Developers
<input type="checkbox"/> AW AWS-Account1-Developers	Group	AzureAD-Developers,AzureAD-Developers

If you can't see or select a role, go back to the **Provisioning** page to confirm successful provisioning in the Microsoft Entra provisioning agent, and make sure the IAM User account has the correct permissions. You can also restart the provisioning engine to attempt the import again:



Test Microsoft Entra SSO into AWS Management Console

Test signing-in as each of the test users to confirm that the SSO works.

1. Launch a new private browser session to ensure that other stored credentials don't conflict with testing.
2. Go to <https://myapps.microsoft.com>, using the Test-AWSAdmin or Test-AWSDeveloper Microsoft Entra user account credentials you created previously.
3. You should see the new icon for the AWS Console app. Select the icon, and follow any authentication prompts:



Contoso-AWS-Ac...

4. Once you're signed into the AWS Console, navigate the features to confirm that this account has the appropriate delegated access.
5. Notice the naming format for the user sign-in session:

ROLE / UPN / AWS Account Number

You can use this user sign-in session information for tracking user sign-in activity in Defender for Cloud Apps or Microsoft Sentinel.

AzureAD-Developers/Richard@developer.com@8818-2327-474

6. Sign out, and repeat the process for the other test user account to confirm the differences in role mapping and permissions.

Enable Conditional Access

To create a new Conditional Access policy that requires MFA:

1. In the Azure portal, navigate to **Microsoft Entra ID > Security**, and then select **Conditional Access**.
2. In the left navigation, select **Policies**.

Policies

New policy | What If | Refresh | Got feedback?

New Conditional Access policies now apply to all client app types when not cor...

Policy Name	State
Exchange Online Requires Compliant Device	Off

3. Select **New policy**, and complete the form as follows:

- **Name:** Enter *AWS Console – MFA*
- **Users and Groups:** Select the two role groups you created earlier:
 - **AWS-Account1-Administrators**
 - **AWS-Account1-Developers**
- **Grant:** Select **Require multi-factor authentication**

4. Set **Enable policy** to **On**.

AWS Console - MFA

Conditional access policy

[Delete](#)

Control user access based on conditional access policy to bring signals together, to make decisions, and enforce organizational policies. [Learn more](#)

Name *

Assignments

Users and groups (1) >
Specific users included

Cloud apps or actions (1) >
1 app included

Conditions (0) >
0 conditions selected

Access controls

Grant (1) >
1 control selected

Session (0) >
0 controls selected

Enable policy
 Report-only On Off

Grant

Control user access enforcement to block or grant access. [Learn more](#)

Block access
 Grant access

Require multi-factor authentication (1)
 Require device to be marked as compliant (1)
 Require Hybrid Azure AD joined device (1)
 Require approved client app (1)
[See list of approved client apps](#)
 Require app protection policy (Preview) (1)
[See list of policy protected client apps](#)
 Require password change (Preview) (1)

For multiple controls

Require all the selected controls
 Require one of the selected controls

⚠ This policy impacts the Azure AD device registration service. So access controls that require device registration are not available.

5. Select **Create**. The policy takes effect immediately.
6. To test the Conditional Access policy, sign out of the testing accounts, open a new in-private browsing session, and sign in with one of the role group accounts. You see the MFA prompt:



test-awsadmin@organization.com

More information required

Your organization needs more information to keep your account secure

[Use a different account](#)

[Learn more](#)

[Next](#)

Contoso

7. Complete the MFA setup process. It's best to use the mobile app for authentication, instead of relying on SMS.

Additional security verification

Secure your account by adding phone verification to your password. [View video](#) to know how to secure your account

Step 1: How should we contact you?

Mobile app



How do you want to use the mobile app?

- Receive notifications for verification
 Use verification code

To use these verification methods, you must set up the Microsoft Authenticator app.

[Set up](#)

Please configure the mobile app.

You might need to create several Conditional Access policies to meet business needs for strong authentication. Consider the naming convention you use when creating the policies to ensure ease of identification and ongoing maintenance. Also, unless MFA is

already widely deployed, make sure the policy is scoped to affect only the intended users. Other policies should cover other user groups' needs.

Once you enable Conditional Access, you can impose other controls such as PAM and just-in-time (JIT) provisioning. For more information, see [What is automated SaaS app user provisioning in Microsoft Entra ID](#).

If you have Defender for Cloud Apps, you can use Conditional Access to configure Defender for Cloud Apps session policies. For more information, see [Configure Microsoft Entra session policies for AWS activities](#).

Next steps

- For security guidance from AWS, see [Best practices for securing AWS accounts and resources](#).
- For the latest Microsoft security information, see www.microsoft.com/security.
- For full details of how to implement and manage Microsoft Entra ID, see [Securing Azure environments with Microsoft Entra ID](#).
- [AWS tutorial: Microsoft Entra ID with IDP SSO](#)
- [Microsoft tutorial: SSO for AWS](#)
- [PIM deployment plan](#)
- [Identity protection security overview](#)
- [What is Microsoft Defender for Identity?](#)
- [Connect AWS to Microsoft Defender for Cloud Apps](#)
- [How Defender for Cloud Apps helps protect your Amazon Web Services \(AWS\) environment](#)

Related resources

- For in-depth coverage and comparison of Azure and AWS features, see the [Azure for AWS professionals](#) content set.
- [Security and identity on Azure and AWS](#)
- [Defender for Cloud Apps and Microsoft Sentinel for AWS](#)

Onboard an Amazon Web Services (AWS) account

Article • 12/20/2023

This article describes how to onboard an Amazon Web Services (AWS) account in Microsoft Entra Permissions Management.

ⓘ Note

You must have Global Administrator permissions to perform the tasks in this article.

Explanation

There are several moving parts across AWS and Azure, which are required to be configured before onboarding.

- A Microsoft Entra OIDC App
- An AWS OIDC account
- An (optional) AWS Management account
- An (optional) AWS Central logging account
- An AWS OIDC role
- An AWS Cross Account role assumed by OIDC role

Onboard an AWS account

1. If the **Data Collectors** dashboard isn't displayed when Permissions Management launches:
 - In the Permissions Management home page, select **Settings** (the gear icon), and then select the **Data Collectors** subtab.
2. On the **Data Collectors** dashboard, select **AWS**, and then select **Create Configuration**.

1. Create a Microsoft Entra OIDC App

1. On the **Permissions Management Onboarding - Microsoft Entra OIDC App Creation** page, enter the **OIDC Azure app name**.

This app is used to set up an OpenID Connect (OIDC) connection to your AWS account. OIDC is an interoperable authentication protocol based on the OAuth 2.0 family of specifications. The scripts generated on this page create the app of this specified name in your Microsoft Entra tenant with the right configuration.

2. To create the app registration, copy the script and run it in your Azure command-line app.

! Note

- a. To confirm that the app was created, open **App registrations** in Azure and, on the **All applications** tab, locate your app.
- b. Select the app name to open the **Expose an API** page. The **Application ID URI** displayed in the **Overview** page is the *audience value* used while making an OIDC connection with your AWS account.

3. Return to Permissions Management, and in the **Permissions Management Onboarding - Microsoft Entra OIDC App Creation**, select **Next**.

2. Set up an AWS OIDC account

1. In the **Permissions Management Onboarding - AWS OIDC Account Setup** page, enter the **AWS OIDC account ID** where the OIDC provider is created. You can change the role name to your requirements.
2. Open another browser window and sign in to the AWS account where you want to create the OIDC provider.
3. Select **Launch Template**. This link takes you to the **AWS CloudFormation create stack** page.
4. Scroll to the bottom of the page, and in the **Capabilities** box, select **I acknowledge that AWS CloudFormation might create IAM resources with custom names**. Then select **Create Stack**.

This AWS CloudFormation stack creates an OIDC Identity Provider (IdP) representing Microsoft Entra STS and an AWS IAM role with a trust policy that allows external identities from Microsoft Entra ID to assume it via the OIDC IdP. These entities are listed on the **Resources** page.

5. Return to Permissions Management, and in the **Permissions Management Onboarding - AWS OIDC Account Setup** page, select **Next**.

3. Set up the AWS Management account connection (Optional)

1. If your organization has Service Control Policies (SCPs) that govern some or all of the member accounts, set up the Management account connection in the **Permissions Management Onboarding - AWS Management Account Details** page.

Setting up the Management account connection allows Permissions Management to auto-detect and onboard any AWS member accounts that have the correct Permissions Management role.

2. In the **Permissions Management Onboarding - AWS Management Account Details** page, enter the **Management Account ID** and **Management Account Role**.
3. Open another browser window and sign in to the AWS console for your Management account.
4. Return to Permissions Management, and in the **Permissions Management Onboarding - AWS Management Account Details** page, select **Launch Template**.

The **AWS CloudFormation create stack** page opens, displaying the template.

5. Review the information in the template, make changes, if necessary, then scroll to the bottom of the page.
6. In the **Capabilities** box, select **I acknowledge that AWS CloudFormation might create IAM resources with custom names**. Then select **Create stack**.

This AWS CloudFormation stack creates a role in the Management account with the necessary permissions (policies) to collect SCPs and list all the accounts in your organization.

A trust policy is set on this role to allow the OIDC role created in your AWS OIDC account to access it. These entities are listed in the **Resources** tab of your CloudFormation stack.

7. Return to Permissions Management, and in **Permissions Management Onboarding - AWS Management Account Details**, select **Next**.

4. Set up the AWS Central logging account connection (Optional but recommended)

1. If your organization has a central logging account where logs from some or all of your AWS account are stored, in the **Permissions Management Onboarding - AWS Central Logging Account Details** page, set up the logging account connection.

In the **Permissions Management Onboarding - AWS Central Logging Account Details** page, enter the **Logging Account ID** and **Logging Account Role**.

2. In another browser window, sign in to the AWS console for the AWS account you use for central logging.
3. Return to Permissions Management, and in the **Permissions Management Onboarding - AWS Central Logging Account Details** page, select **Launch Template**.

The **AWS CloudFormation create stack** page opens, displaying the template.

4. Review the information in the template, make changes, if necessary, then scroll to the bottom of the page.
5. In the **Capabilities** box, select **I acknowledge that AWS CloudFormation might create IAM resources with custom names**, and then select **Create stack**.

This AWS CloudFormation stack creates a role in the logging account with the necessary permissions (policies) to read S3 buckets used for central logging. A trust policy is set on this role to allow the OIDC role created in your AWS OIDC account to access it. These entities are listed in the **Resources** tab of your CloudFormation stack.

6. Return to Permissions Management, and in the **Permissions Management Onboarding - AWS Central Logging Account Details** page, select **Next**.

5. Set up an AWS member account

Select **Enable AWS SSO checkbox**, if the AWS account access is configured through AWS SSO.

Choose from three options to manage AWS accounts.

Option 1: Automatically manage

Choose this option to automatically detect and add to the monitored account list, without extra configuration. Steps to detect list of accounts and onboard for collection:

- Deploy Management account CFT (Cloudformation template) which creates organization account role that grants permission to OIDC role created earlier to list accounts, OUs and SCPs.
- If AWS SSO is enabled, organization account CFT also adds policy needed to collect AWS SSO configuration details.
- Deploy Member account CFT in all the accounts that need to be monitored by Microsoft Entra Permissions Management. These actions create a cross account role that trusts the OIDC role created earlier. The SecurityAudit policy is attached to the role created for data collection.

Any current or future accounts found get onboarded automatically.

To view status of onboarding after saving the configuration:

- Go to **Data Collectors** tab.
- Click on the status of the data collector.
- View accounts on the **In Progress** page

Option 2: Enter authorization systems

1. In the **Permissions Management Onboarding - AWS Member Account Details** page, enter the **Member Account Role** and the **Member Account IDs**.

You can enter up to 100 account IDs. Click the plus icon next to the text box to add more account IDs.

! Note

Do the following steps for each account ID you add:

2. Open another browser window and sign in to the AWS console for the member account.
3. Return to the **Permissions Management Onboarding - AWS Member Account Details** page, select **Launch Template**.

The **AWS CloudFormation create stack** page opens, displaying the template.

4. In the **CloudTrailBucketName** page, enter a name.

You can copy and paste the **CloudTrailBucketName** name from the **Trails** page in AWS.

Note

A *cloud bucket* collects all the activity in a single account that Permissions Management monitors. Enter the name of a cloud bucket here to provide Permissions Management with the access required to collect activity data.

5. From the **Enable Controller** dropdown, select:

- **True**, if you want the controller to provide Permissions Management with read and write access so that any remediation you want to do from the Permissions Management platform can be done automatically.
- **False**, if you want the controller to provide Permissions Management with read-only access.

6. Scroll to the bottom of the page, and in the **Capabilities** box, select **I acknowledge that AWS CloudFormation might create IAM resources with custom names**. Then select **Create stack**.

This AWS CloudFormation stack creates a collection role in the member account with necessary permissions (policies) for data collection.

A trust policy is set on this role to allow the OIDC role created in your AWS OIDC account to access it. These entities are listed in the **Resources** tab of your CloudFormation stack.

7. Return to Permissions Management, and in the **Permissions Management Onboarding - AWS Member Account Details** page, select **Next**.

This step completes the sequence of required connections from Microsoft Entra STS to the OIDC connection account and the AWS member account.

Option 3: Select authorization systems

This option detects all AWS accounts that are accessible through OIDC role access created earlier.

- Deploy Management account CFT (Cloudformation template) which creates organization account role that grants permission to OIDC role created earlier to list accounts, OUs and SCPs.
- If AWS SSO is enabled, organization account CFT also adds policy needed to collect AWS SSO configuration details.

- Deploy Member account CFT in all the accounts that need to be monitored by Microsoft Entra Permissions Management. These actions create a cross account role that trusts the OIDC role created earlier. The SecurityAudit policy is attached to the role created for data collection.
- Click Verify and Save.
- Go to the newly create Data Collector row under AWSdata collectors.
- Click on Status column when the row has **Pending** status
- To onboard and start collection, choose specific ones from the detected list and consent for collection.

6. Review and save

1. In **Permissions Management Onboarding – Summary**, review the information you've added, and then select **Verify Now & Save**.

The following message appears: **Successfully created configuration.**

On the **Data Collectors** dashboard, the **Recently Uploaded On** column displays **Collecting**. The **Recently Transformed On** column displays **Processing**.

The status column in your Permissions Management UI shows you which step of data collection you're at:

- **Pending**: Permissions Management has not started detecting or onboarding yet.
- **Discovering**: Permissions Management is detecting the authorization systems.
- **In progress**: Permissions Management has finished detecting the authorization systems and is onboarding.
- **Onboarded**: Data collection is complete, and all detected authorization systems are onboarded to Permissions Management.

7. View the data

1. To view the data, select the **Authorization Systems** tab.

The **Status** column in the table displays **Collecting Data**.

The data collection process takes some time and occurs in approximately 4-5 hour intervals in most cases. The time frame depends on the size of the authorization system you have and how much data is available for collection.

Next steps

- For information on how to onboard a Microsoft Azure subscription, see [Onboard a Microsoft Azure subscription](#).
- For information on how to onboard a Google Cloud Platform (GCP) project, see [Onboard a Google Cloud Platform \(GCP\) project](#).
- For information on how to enable or disable the controller after onboarding is complete, see [Enable or disable the controller](#).
- For information on how to add an account/subscription/project after onboarding is complete, see [Add an account/subscription/project after onboarding is complete](#).

Tutorial: Microsoft Entra single sign-on (SSO) integration with Amazon Managed Grafana

Article • 10/23/2023

In this tutorial, you'll learn how to integrate Amazon Managed Grafana with Microsoft Entra ID. When you integrate Amazon Managed Grafana with Microsoft Entra ID, you can:

- Control in Microsoft Entra ID who has access to Amazon Managed Grafana.
- Enable your users to be automatically signed-in to Amazon Managed Grafana with their Microsoft Entra accounts.
- Manage your accounts in one central location.

Prerequisites

To get started, you need the following items:

- A Microsoft Entra subscription. If you don't have a subscription, you can get a [free account](#).
- Amazon Web Services (AWS) [free account](#).
- Amazon Managed Grafana single sign-on (SSO) enabled subscription.

Scenario description

In this tutorial, you configure and test Microsoft Entra SSO in a test environment.

- Amazon Managed Grafana supports **SP** initiated SSO.
- Amazon Managed Grafana supports **Just In Time** user provisioning.

Add Amazon Managed Grafana from the gallery

To configure the integration of Amazon Managed Grafana into Microsoft Entra ID, you need to add Amazon Managed Grafana from the gallery to your list of managed SaaS apps.

1. Sign in to the Microsoft Entra admin center [↗](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > New application**.
3. In the **Add from the gallery** section, type **Amazon Managed Grafana** in the search box.
4. Select **Amazon Managed Grafana** from results panel and then add the app. Wait a few seconds while the app is added to your tenant.

Alternatively, you can also use the [Enterprise App Configuration Wizard ↗](#). In this wizard, you can add an application to your tenant, add users/groups to the app, assign roles, as well as walk through the SSO configuration as well. [Learn more about Microsoft 365 wizards](#).

Configure and test Microsoft Entra SSO for Amazon Managed Grafana

Configure and test Microsoft Entra SSO with Amazon Managed Grafana using a test user called **B.Simon**. For SSO to work, you need to establish a link relationship between a Microsoft Entra user and the related user in Amazon Managed Grafana.

To configure and test Microsoft Entra SSO with Amazon Managed Grafana, perform the following steps:

1. [Configure Microsoft Entra SSO](#) - to enable your users to use this feature.
 - a. [Create a Microsoft Entra test user](#) - to test Microsoft Entra single sign-on with B.Simon.
 - b. [Assign the Microsoft Entra test user](#) - to enable B.Simon to use Microsoft Entra single sign-on.
2. [Configure Amazon Managed Grafana SSO](#) - to configure the single sign-on settings on application side.
 - a. [Create Amazon Managed Grafana test user](#) - to have a counterpart of B.Simon in Amazon Managed Grafana that is linked to the Microsoft Entra representation of user.
3. [Test SSO](#) - to verify whether the configuration works.

Configure Microsoft Entra SSO

Follow these steps to enable Microsoft Entra SSO.

1. Sign in to the Microsoft Entra admin center [↗](#) as at least a [Cloud Application Administrator](#).

2. Browse to **Identity > Applications > Enterprise applications > Amazon Managed Grafana > Single sign-on.**
3. On the **Select a single sign-on method** page, select **SAML**.
4. On the **Set up single sign-on with SAML** page, click the pencil icon for **Basic SAML Configuration** to edit the settings.

Set up Single Sign-On with SAML

An SSO implementation based on federation protocols improves security, reliability, and end user experiences and is easier to implement. Choose SAML single sign-on whenever possible for existing applications that do not use OpenID Connect or OAuth. [Learn more.](#)

Read the [configuration guide](#) for help integrating <App Name>

1

Basic SAML Configuration

 Edit

Identifier (Entity ID)
Reply URL (Assertion Consumer Service URL)
Sign on URL
Relay State (Optional)
Logout Url (Optional)

5. On the **Basic SAML Configuration** section, perform the following steps:

- a. In the **Identifier (Entity ID)** text box, type a URL using the following pattern:

`https://<namespace>.grafana-workspace.<region>.amazonaws.com/saml/metadata`

- b. In the **Sign on URL** text box, type a URL using the following pattern:

`https://<namespace>.grafana-workspace.<region>.amazonaws.com/login/saml`

Note

These values are not real. Update these values with the actual Identifier and Sign on URL. Contact [Amazon Managed Grafana Client support team](#) to get these values. You can also refer to the patterns shown in the **Basic SAML Configuration** section.

6. Amazon Managed Grafana application expects the SAML assertions in a specific format, which requires you to add custom attribute mappings to your SAML token attributes configuration. The following screenshot shows the list of default attributes.

2

Attributes & Claims	
givenname	user.givenname
surname	user.surname
emailaddress	user.mail
name	user.userprincipalname
Unique User Identifier	user.userprincipalname

7. In addition to above, Amazon Managed Grafana application expects few more attributes to be passed back in SAML response which are shown below. These attributes are also pre populated but you can review them as per your requirements.

Name	Source attribute
displayName	user.displayname
mail	user.userprincipalname

8. On the **Set up single sign-on with SAML** page, in the **SAML Signing Certificate** section, find **Federation Metadata XML** and select **Download** to download the certificate and save it on your computer.

3

SAML Signing Certificate	
Status	Active
Thumbprint	<Thumbprintvalue>
Expiration	<Expiration>
Notification Email	<Email address>
App Federation Metadata Url	<App Federation Metadata Url>
Certificate (Base64)	Download
Certificate (Raw)	Download
Federation Metadata XML	Download

9. On the **Set up Amazon Managed Grafana** section, copy the appropriate URL(s) based on your requirement.

Set up <Application Name>	
You'll need to configure the application to link with Microsoft Entra ID.	
Login URL	https://login.microsoftonline.com/4f74... 
Microsoft Entra ID Identifier	https://sts.windows.net/4f7437a6-3d76... 
Logout URL	https://login.microsoftonline.com/4f74... 

Create a Microsoft Entra test user

In this section, you'll create a test user called B.Simon.

1. Sign in to the Microsoft Entra admin center [↗](#) as at least a [User Administrator](#).
2. Browse to **Identity > Users > All users**.
3. Select **New user > Create new user**, at the top of the screen.
4. In the **User** properties, follow these steps:
 - a. In the **Display name** field, enter **B.Simon**.
 - b. In the **User principal name** field, enter the **username@companydomain.extension**. For example, **B.Simon@contoso.com**.
 - c. Select the **Show password** check box, and then write down the value that's displayed in the **Password** box.
 - d. Select **Review + create**.
5. Select **Create**.

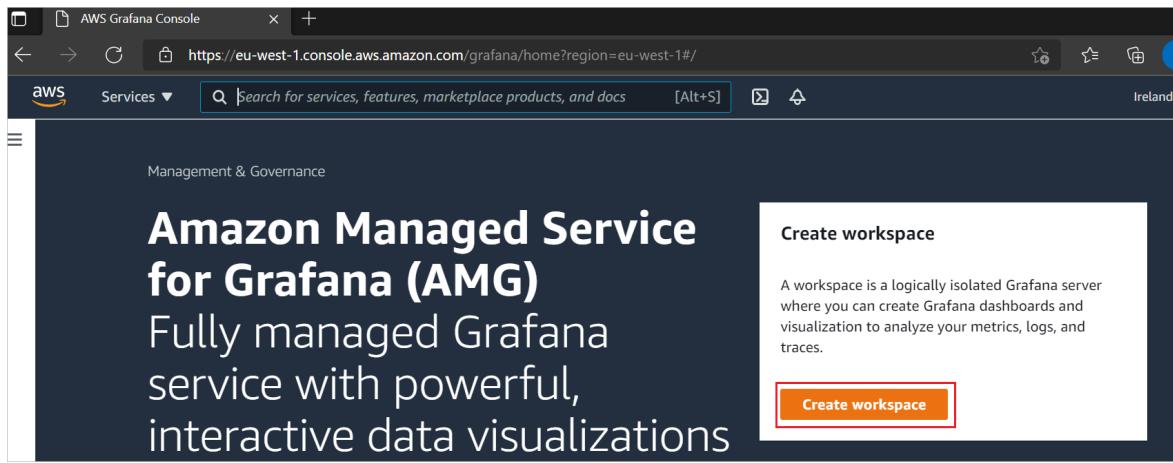
Assign the Microsoft Entra test user

In this section, you'll enable B.Simon to use single sign-on by granting access to Amazon Managed Grafana.

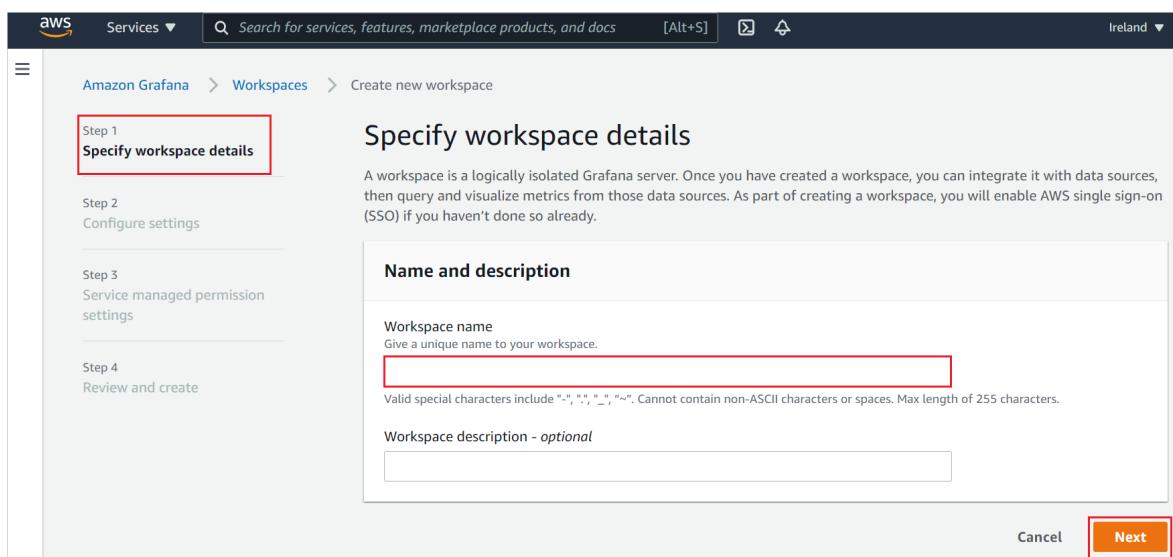
1. Sign in to the Microsoft Entra admin center [↗](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > Amazon Managed Grafana**.
3. In the app's overview page, select **Users and groups**.
4. Select **Add user/group**, then select **Users and groups** in the **Add Assignment** dialog.
 - a. In the **Users and groups** dialog, select **B.Simon** from the **Users** list, then click the **Select** button at the bottom of the screen.
 - b. If you are expecting a role to be assigned to the users, you can select it from the **Select a role** dropdown. If no role has been set up for this app, you see "Default Access" role selected.
 - c. In the **Add Assignment** dialog, click the **Assign** button.

Configure Amazon Managed Grafana SSO

1. Log in to your Amazon Managed Grafana Console as an administrator.
2. Click **Create workspace**.



3. In the **Specify workspace details** page, type a unique **Workspace name** and click **Next**.



4. In the **Configure settings** page, select **Security Assertion Markup Language(SAML)** checkbox and enable **Service managed** as permission type and click **Next**.

The screenshot shows the 'Configure settings' step of the workspace creation process. On the left, a sidebar lists steps: Step 1 (Specify workspace details), Step 2 (Configure settings, highlighted with a red box), Step 3 (Service managed permission settings), and Step 4 (Review and create). The main content area is titled 'Configure settings' with an 'Info' link. It contains two sections: 'Authentication access' and 'Permission type'. In 'Authentication access', the 'Choose at least one authentication method.' section includes options for AWS Single Sign-On (AWS SSO) and Security Assertion Markup Language (SAML), with SAML selected (indicated by a checked checkbox and a red box). A note states: 'You will need to complete additional steps to finish SAML configuration once this workspace is created.' Below this is a note: '(Pending user input after workspace creation)'. In the 'Permission type' section, 'Service managed' is selected (indicated by a blue radio button and a red box), with the note: 'We will automatically provision the permissions for you based on the AWS services you choose in the next step.' The 'Customer managed' option is also shown. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' highlighted by a red box.

5. In the **Service managed permission settings**, select **Current account** and click **Next**.

The screenshot shows the 'Service managed permission settings' step of the workspace creation process. The sidebar shows steps: Step 1 (Specify workspace details), Step 2 (Configure settings), Step 3 (Service managed permission settings, highlighted with a red box), and Step 4 (Review and create). The main content area is titled 'Service managed permission settings' with an 'Info' link. It contains two sections: 'IAM permission access settings' and 'Data sources and notification channels - optional'. In 'IAM permission access settings', 'Current account' is selected (indicated by a blue radio button and a red box), with the note: 'Use Grafana to monitor resources in your current account.' The 'Organization' option is also shown. In 'Data sources and notification channels - optional', there are sections for 'Data sources' and 'Notification channels'. Under 'Data sources', several AWS services are listed with checkboxes: AWS IoT SiteWise, AWS X-Ray, Amazon CloudWatch, Amazon Elasticsearch Service, Amazon Managed Service for Prometheus, and Amazon TimeStream. Under 'Notification channels', the Amazon SNS service is listed with a checkbox. At the bottom right are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' highlighted by a red box.

6. In the **Review and create** page, verify all the workspace details and click **Create workspace**.

aws Services ▾ Search for services, features, marketplace products, and docs [Alt+S] Ireland ▾

Amazon Grafana > Workspaces > Create new workspace

Step 1
Specify workspace details

Step 2
Configure settings

Step 3
Service managed permission settings

Step 4
Review and create

Review and create

Step 1: Specify workspace details

Name and description

Workspace name Workspace description

Step 2: Configure settings

Authentication access

AWS Single Sign-On (SSO) Disabled Security Assertion Markup Language (SAML) Pending user input after workspace creation

Permission type

Permission Service managed

Step 3: Automatic permission settings

IAM permission access settings

Account access specified Current account

Data sources and notification channels

Data sources Notification channels

Cancel Previous **Create workspace**

7. After creating workspace, click **Complete setup** to complete the SAML configuration.

aws Services ▾ Search for services, features, marketplace products, and docs [Alt+S] Ireland ▾

Security Assertion Markup Language (SAML) setup pending user input.
You need to complete additional steps to finish the SAML configuration. **Complete setup**

testssso

Summary	Info
Description <input type="button" value="Edit"/>	Date created 2021-08-09
Grafana workspace URL <workspace URL>	Authentication access ⚠ SAML
Status Active	IAM role <input type="button" value="Edit"/> Enterprise license Upgrade to Grafana Enterprise
	Grafana version 7.5

8. In the **Security Assertion Markup Language(SAML)** page, perform the following steps.

Configure your IdP

Configure SAML support on your IdP for this workspace. Provide the ID and the URL information below to your IdP. This causes IdP metadata to be generated.

Service provider identifier (Entity ID)

Service provider login URL

Service provider reply URL (Assertion consumer service URL)

Import the metadata

Once your IdP is configured, an IdP metadata is generated. Import the metadata from an XML file, specify a URL or copy and paste to the editor below.

Import method

URL
Specify a URL and we will copy the metadata.

Upload or copy/paste
Upload the XML file from your local computer or copy/paste.

Import the metadata

You can copy and paste the metadata here, instead of uploading a file

.xml file supported

Assertion mapping Info

Configure SAML assertion attributes to map your IdP user information to AMG workspace users as well as assign orgs and users access to the workspace.

Assertion attribute role for admin
An admin is required to set up data sources, assign user permissions, and more.

Assertion attribute role

Admin role values

Enter comma separated values for multiple roles.

I want to opt-out of assigning admins to my workspace.

- a. Copy **Service provider identifier(Entity ID)** value, paste this value into the **Identifier** text box in the **Basic SAML Configuration** section.
- b. Copy **Service provider reply URL(Assertion consumer service URL)** value, paste this value into the **Reply URL** text box in the **Basic SAML Configuration** section.
- c. Copy **Service provider login URL** value, paste this value into the **Sign on URL** text box in the **Basic SAML Configuration** section.
- d. Open the downloaded **Federation Metadata XML** into Notepad and upload the XML file by clicking **Choose file** option.
- e. In the **Assertion mapping** section, fill the required values according to your requirement.
- f. Click **Save SAML configuration**.

Create Amazon Managed Grafana test user

In this section, a user called Britta Simon is created in Amazon Managed Grafana. Amazon Managed Grafana supports just-in-time user provisioning, which is enabled by

default. There is no action item for you in this section. If a user doesn't already exist in Amazon Managed Grafana, a new one is created after authentication.

Test SSO

In this section, you test your Microsoft Entra single sign-on configuration with following options.

- Click on **Test this application**, this will redirect to Amazon Managed Grafana Sign-on URL where you can initiate the login flow.
- Go to Amazon Managed Grafana Sign-on URL directly and initiate the login flow from there.
- You can use Microsoft My Apps. When you click the Amazon Managed Grafana tile in the My Apps, this will redirect to Amazon Managed Grafana Sign-on URL. For more information about the My Apps, see [Introduction to the My Apps](#).

Next steps

Once you configure Amazon Managed Grafana you can enforce session control, which protects exfiltration and infiltration of your organization's sensitive data in real time. Session control extends from Conditional Access. [Learn how to enforce session control with Microsoft Defender for Cloud Apps](#).

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) | [Get help at Microsoft Q&A](#)

Tutorial: Microsoft Entra SSO integration with AWS Single-Account Access

Article • 11/01/2023

In this tutorial, you'll learn how to integrate AWS Single-Account Access with Microsoft Entra ID. When you integrate AWS Single-Account Access with Microsoft Entra ID, you can:

- Control in Microsoft Entra ID who has access to AWS Single-Account Access.
- Enable your users to be automatically signed-in to AWS Single-Account Access with their Microsoft Entra accounts.
- Manage your accounts in one central location.

Understanding the different AWS applications in the Microsoft Entra application gallery

Use the information below to make a decision between using the AWS Single Sign-On and AWS Single-Account Access applications in the Microsoft Entra application gallery.

AWS Single Sign-On

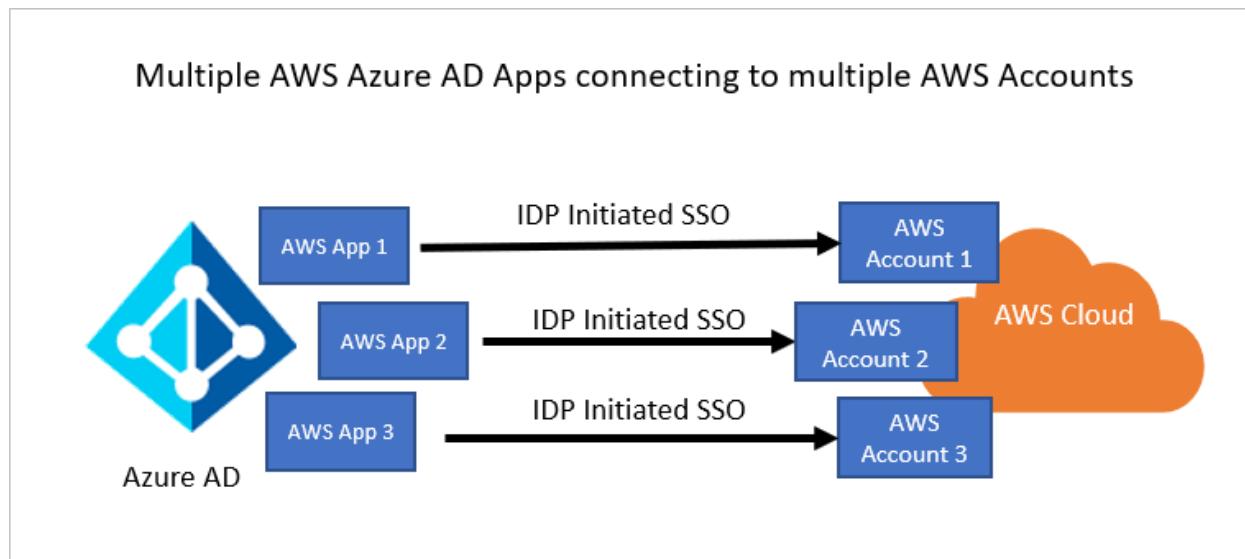
[AWS Single Sign-On](#) was added to the Microsoft Entra application gallery in February 2021. It makes it easy to manage access centrally to multiple AWS accounts and AWS applications, with sign-in through Microsoft Entra ID. Federate Microsoft Entra ID with AWS SSO once, and use AWS SSO to manage permissions across all of your AWS accounts from one place. AWS SSO provisions permissions automatically and keeps them current as you update policies and access assignments. End users can authenticate with their Microsoft Entra credentials to access the AWS Console, Command Line Interface, and AWS SSO integrated applications.

AWS Single-Account Access

[AWS Single-Account Access](#) has been used by customers over the past several years and enables you to federate Microsoft Entra ID to a single AWS account and use Microsoft Entra ID to manage access to AWS IAM roles. AWS IAM administrators define roles and policies in each AWS account. For each AWS account, Microsoft Entra administrators federate to AWS IAM, assign users or groups to the account, and configure Microsoft Entra ID to send assertions that authorize role access.

Feature	AWS Single Sign-On	AWS Single-Account Access
Conditional Access	Supports a single Conditional Access policy for all AWS accounts.	Supports a single Conditional Access policy for all accounts or custom policies per account
CLI access	Supported	Supported
Privileged Identity Management	Public preview	Not yet supported
Centralize account management	Centralize account management in AWS.	Centralize account management in Microsoft Entra ID (will likely require a Microsoft Entra enterprise application per account).
SAML certificate	Single certificate	Separate certificates per app / account

AWS Single-Account Access architecture



You can configure multiple identifiers for multiple instances. For example:

- `https://signin.aws.amazon.com/saml#1`
- `https://signin.aws.amazon.com/saml#2`

With these values, Microsoft Entra ID removes the value of #, and sends the correct value `https://signin.aws.amazon.com/saml` as the audience URL in the SAML token.

We recommend this approach for the following reasons:

- Each application provides you with a unique X509 certificate. Each instance of an AWS app instance can then have a different certificate expiry date, which can be

managed on an individual AWS account basis. Overall certificate rollover is easier in this case.

- You can enable user provisioning with an AWS app in Microsoft Entra ID, and then our service fetches all the roles from that AWS account. You don't have to manually add or update the AWS roles on the app.
- You can assign the app owner individually for the app. This person can manage the app directly in Microsoft Entra ID.

 **Note**

Make sure you use a gallery application only.

Prerequisites

To get started, you need the following items:

- A Microsoft Entra subscription. If you don't have a subscription, you can get a [free account](#).
- An AWS IAM IdP enabled subscription.
- Along with Cloud Application Administrator, Application Administrator can also add or manage applications in Microsoft Entra ID. For more information, see [Azure built-in roles](#).

 **Note**

Roles should not be manually edited in Microsoft Entra ID when doing role imports.

Scenario description

In this tutorial, you configure and test Microsoft Entra SSO in a test environment.

- AWS Single-Account Access supports SP and IDP initiated SSO.

 **Note**

Identifier of this application is a fixed string value so only one instance can be configured in one tenant.

Adding AWS Single-Account Access from the gallery

To configure the integration of AWS Single-Account Access into Microsoft Entra ID, you need to add AWS Single-Account Access from the gallery to your list of managed SaaS apps.

1. Sign in to the [Microsoft Entra admin center](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > New application**.
3. In the **Add from the gallery** section, type **AWS Single-Account Access** in the search box.
4. Select **AWS Single-Account Access** from results panel and then add the app. Wait a few seconds while the app is added to your tenant.

Alternatively, you can also use the [Enterprise App Configuration Wizard](#). In this wizard, you can add an application to your tenant, add users/groups to the app, assign roles, as well as walk through the SSO configuration as well. [Learn more about Microsoft 365 wizards](#).

Alternatively, you can also use the [Enterprise App Configuration Wizard](#). In this wizard, you can add an application to your tenant, add users/groups to the app, assign roles, as well as walk through the SSO configuration as well. You can learn more about O365 wizards [here](#).

Configure and test Microsoft Entra SSO for AWS Single-Account Access

Configure and test Microsoft Entra SSO with AWS Single-Account Access using a test user called **B.Simon**. For SSO to work, you need to establish a link relationship between a Microsoft Entra user and the related user in AWS Single-Account Access.

To configure and test Microsoft Entra SSO with AWS Single-Account Access, perform the following steps:

1. **Configure Microsoft Entra SSO** - to enable your users to use this feature.
 - a. **Create a Microsoft Entra test user** - to test Microsoft Entra single sign-on with B.Simon.
 - b. **Assign the Microsoft Entra test user** - to enable B.Simon to use Microsoft Entra single sign-on.

2. **Configure AWS Single-Account Access SSO** - to configure the single sign-on settings on application side.
 - a. [Create AWS Single-Account Access test user](#) - to have a counterpart of B.Simon in AWS Single-Account Access that is linked to the Microsoft Entra representation of user.
 - b. [How to configure role provisioning in AWS Single-Account Access](#)
3. [Test SSO](#) - to verify whether the configuration works.

Configure Microsoft Entra SSO

Follow these steps to enable Microsoft Entra SSO.

1. Sign in to the [Microsoft Entra admin center](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > AWS Single-Account Access > Single sign-on**.
3. On the **Select a single sign-on method** page, select **SAML**.
4. On the **Set up single sign-on with SAML** page, click the pencil icon for **Basic SAML Configuration** to edit the settings.

Set up Single Sign-On with SAML

An SSO implementation based on federation protocols improves security, reliability, and end user experiences and is easier to implement. Choose SAML single sign-on whenever possible for existing applications that do not use OpenID Connect or OAuth. [Learn more](#).

Read the [configuration guide](#) for help integrating <App Name>

1	Basic SAML Configuration <div style="float: right; margin-top: -20px;"></div> <ul style="list-style-type: none"> Identifier (Entity ID) Reply URL (Assertion Consumer Service URL) Sign on URL Relay State (Optional) Logout Url (Optional)
---	--

5. In the **Basic SAML Configuration** section, update both **Identifier (Entity ID)** and **Reply URL** with the same default value: `https://signin.aws.amazon.com/saml`. You must select **Save** to save the configuration changes.
6. When you are configuring more than one instance, provide an identifier value. From second instance onwards, use the following format, including a # sign to specify a unique SPN value.

`https://signin.aws.amazon.com/saml#2`

7. AWS application expects the SAML assertions in a specific format, which requires you to add custom attribute mappings to your SAML token attributes configuration. The following screenshot shows the list of default attributes.

Attributes & Claims		Edit
givenname	user.givenname	
surname	user.surname	
emailaddress	user.mail	
name	user.userprincipalname	
Unique User Identifier	user.userprincipalname	

8. In addition to above, AWS application expects few more attributes to be passed back in SAML response which are shown below. These attributes are also pre populated but you can review them as per your requirements.

Name	Source attribute	Namespace
RoleSessionName	user.userprincipalname	https://aws.amazon.com/SAML/Attributes
Role	user.assignedroles	https://aws.amazon.com/SAML/Attributes
SessionDuration	user.sessionduration	https://aws.amazon.com/SAML/Attributes

Note

AWS expects roles for users assigned to the application. Please set up these roles in Microsoft Entra ID so that users can be assigned the appropriate roles. To understand how to configure roles in Microsoft Entra ID, see [here](#)

9. On the **Set up single sign-on with SAML** page, in the **SAML Signing Certificate** (Step 3) dialog box, select **Add a certificate**.



10. Generate a new SAML signing certificate, and then select **New Certificate**. Enter an email address for certificate notifications.

SAML Signing Certificate

Manage the certificate used by Microsoft Entra ID to sign SAML tokens issued to your app

Save **+ New Certificate** Import Certificate Got feedback?

Status	Expiration Date	Thumbprint
Signing Option	Sign SAML assertion	▼
Signing Algorithm	SHA-256	▼
Notification Email Addresses		

11. In the **SAML Signing Certificate** section, find **Federation Metadata XML** and select **Download** to download the certificate and save it on your computer.

3 SAML Signing Certificate

No certificates exist for this application **Add a certificate** Click Add a certificate

SAML Signing Certificate

Manage the certificate used by Azure AD to sign SAML tokens issued to your app

Save **+ New Certificate** Click New Certificate

Status	Expiration Date	Thumbprint
n/a	06/30/2023	Will be displayed on save
Signing Option	Sign SAML assertion	Click Make certificate active Make certificate active
Signing Algorithm	SHA-256	Base64 certificate download PEM certificate download Raw certificate download
Notification Email Addresses		

3 SAML Signing Certificate

Status	Active
Thumbprint	<Thumbprint Value>
Expiration	<Certificate Expire Date>
Notification Email	<User Notification Email>
App Federation Metadata Url	https://login.microsoftonline.com/4f7437a6-3d76...
Certificate (Base64)	Download
Certificate (Raw)	Download
Federation Metadata XML	Download

12. In the **Set up AWS Single-Account Access** section, copy the appropriate URL(s) based on your requirement.

Set up <Application Name>

You'll need to configure the application to link with Microsoft Entra ID.

Login URL	https://login.microsoftonline.com/4f7437a6-3d76-4...
Microsoft Entra ID Identifier	https://sts.windows.net/4f7437a6-3d76-4...
Logout URL	https://login.microsoftonline.com/4f7437a6-3d76-4...

Create a Microsoft Entra test user

In this section, you'll create a test user called B.Simon.

1. Sign in to the [Microsoft Entra admin center](#) as at least a **User Administrator**.
2. Browse to **Identity > Users > All users**.
3. Select **New user > Create new user**, at the top of the screen.
4. In the **User properties**, follow these steps:
 - a. In the **Display name** field, enter **B.Simon**.
 - b. In the **User principal name** field, enter the **username@companydomain.extension**. For example, **B.Simon@contoso.com**.
 - c. Select the **Show password** check box, and then write down the value that's displayed in the **Password** box.
 - d. Select **Review + create**.
5. Select **Create**.

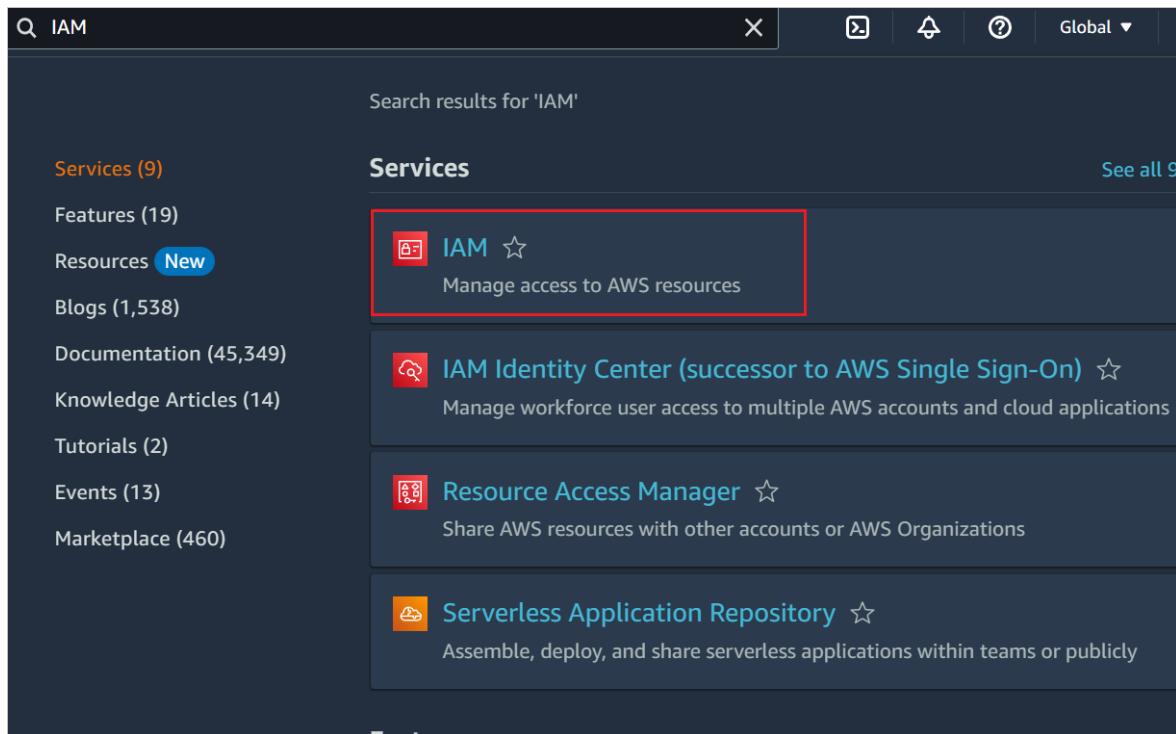
Assign the Microsoft Entra test user

In this section, you'll enable B.Simon to use single sign-on by granting access to AWS Single-Account Access.

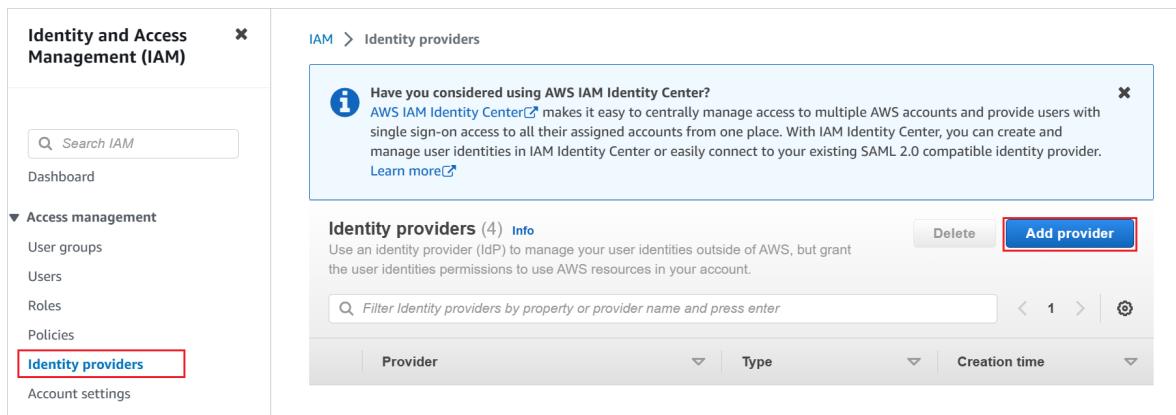
1. Sign in to the [Microsoft Entra admin center](#) as at least a **Cloud Application Administrator**.
2. Browse to **Identity > Applications > Enterprise applications > AWS Single-Account Access**.
3. In the app's overview page, select **Users and groups**.
4. Select **Add user/group**, then select **Users and groups** in the **Add Assignment** dialog.
 - a. In the **Users and groups** dialog, select **B.Simon** from the **Users** list, then click the **Select** button at the bottom of the screen.
 - b. If you are expecting a role to be assigned to the users, you can select it from the **Select a role** dropdown. If no role has been set up for this app, you see "Default Access" role selected.
 - c. In the **Add Assignment** dialog, click the **Assign** button.

Configure AWS Single-Account Access SSO

1. In a different browser window, sign-on to your AWS company site as an administrator.
2. In AWS home page, search for **IAM** and click it.



3. Go to Access management -> Identity Providers and click Add provider button.



4. In the Add an Identity provider page, perform the following steps:

Add an Identity provider

Configure provider

Provider type [Info](#)

SAML

Establish trust between your AWS account and a SAML 2.0 compatible Identity Provider such as Shibboleth or Active Directory Federation Services.

OpenID Connect

Establish trust between your AWS account and Identity Provider services, such as Google or Salesforce.

Provider name

Enter a meaningful name to identify this provider

Maximum 128 characters. Use alphanumeric or '-' characters.

Metadata document [Info](#)

This document is issued by your IdP.

Choose file

File needs to be a valid UTF-8 XML document.

Add tags - optional [Info](#)

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add tag

You can add up to 50 more tags.

[Cancel](#)

Add provider

a. For **Provider type**, select **SAML**.

b. For **Provider name**, type a provider name (for example: *WAAD*).

c. To upload your downloaded **metadata file**, select **Choose file**.

d. Click **Add provider**.

5. Select Roles > Create role.

The screenshot shows the AWS IAM Roles page. On the left, the navigation menu for 'Identity and Access Management (IAM)' is visible, with 'Roles' selected. The main pane displays a list of roles with 40 items. At the top right of the main pane, there is a blue button labeled 'Create role' with a red box drawn around it, indicating it is the target of step 6.

6. On the **Create role** page, perform the following steps:

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity [Info](#)

Trusted entity type

AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.

SAML 2.0 federation Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

SAML 2.0-based provider [Create new](#)

Allow programmatic access only

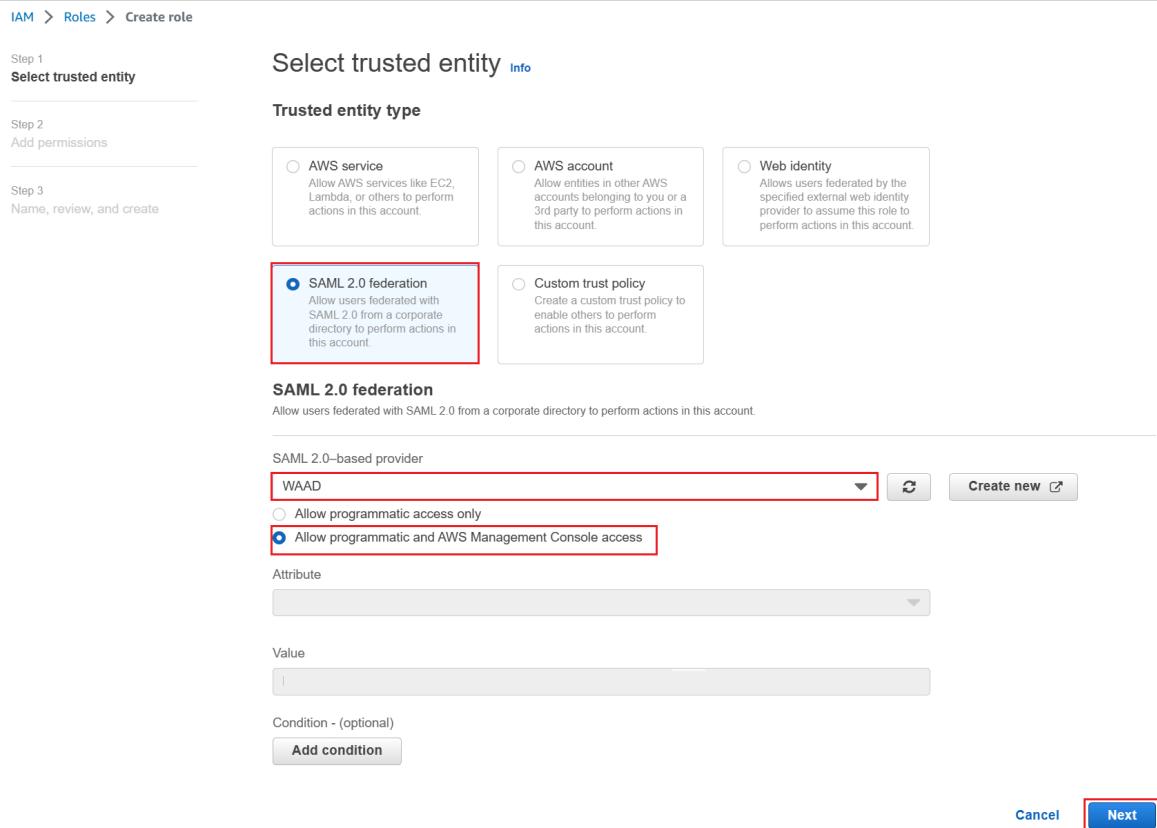
Allow programmatic and AWS Management Console access

Attribute

Value

Condition - (optional) [Add condition](#)

[Cancel](#) [Next](#)



- a. Choose **Trusted entity type**, select **SAML 2.0 federation**.
 - b. Under **SAML 2.0 based provider**, select the **SAML provider** you created previously (for example: **WAAD**).
 - c. Select **Allow programmatic and AWS Management Console access**.
 - d. Select **Next**.
7. On the **Permissions policies** dialog box, attach the appropriate policy, per your organization. Then select **Next**.

Step 1
[Select trusted entity](#)

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions [Info](#)

Permissions policies (880) [Info](#)

Choose one or more policies to attach to your new role.

[Create policy](#)

Filter policies by property or policy name and press enter. 1 2 3 4 5 6 7 ... 44 >

<input type="checkbox"/>	Policy name	Type	Description
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch
<input type="checkbox"/>	AmazonGrafanaAmazo...	Custom...	Allows Amazon Grafana to access Amazon Elasticsearch

► Set permissions boundary - optional [Info](#)

Select a permissions boundary to control the maximum permissions this role can have. This is not a common setting, but you can use it to delegate permission management to others.

[Cancel](#) [Previous](#) **Next**

8. On the **Review** dialog box, perform the following steps:

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role
[Redacted]

Maximum 64 characters. Use alphanumeric and '+_-' characters.

Description
Add a short explanation for this role.
[Redacted]

Maximum 1000 characters. Use alphanumeric and '+_-' characters.

Step 1: Select trusted entities Edit

```

1 "Version": "2012-10-17",
2   "Statement": [
3     {
4       "Effect": "Allow",
5       "Action": "sts:AssumeRoleWithSAML",
6       "Principal": {
7         "Federated": "arn:aws:iam::<Account_ID>:saml-provider/WAAD"
8       },
9       "Condition": {
10         "StringEquals": {
11           "SAML:aud": [
12             "https://signin.aws.amazon.com/saml"
13           ]
14         }
15       }
16     }
17   ]
18 ]
19 
```

Step 2: Add permissions Edit

Permissions policy summary

Policy name	Type	Attached as

Tags

Add tags - optional info
Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add tag
You can add up to 50 more tags.

Cancel Previous Create role

- a. In **Role name**, enter your role name.
 - b. In **Description**, enter the role description.
 - c. Select **Create role**.
 - d. Create as many roles as needed and map them to the identity provider.
9. Use AWS service account credentials for fetching the roles from the AWS account in Microsoft Entra user provisioning. For this, open the AWS console home.
10. In the IAM section, select **Policies** and click **Create policy**.

The screenshot shows the AWS Identity and Access Management (IAM) Policies page. On the left, there's a sidebar with 'Identity and Access Management (IAM)' at the top, followed by 'Dashboard', 'Access management' (with 'User groups', 'Users', 'Roles', and 'Policies' listed), 'Identity providers', and 'Account settings'. The 'Policies' link is highlighted with a red box. The main area shows a table of policies with columns: Policy name, Type, Used as, and Description. There are four entries, all of which are 'Customer managed' and 'Permissions policy (1)', allowing access to Amazon Grafana. A search bar at the top says 'Filter policies by property or policy name and press enter.' and a navigation bar at the bottom shows pages 1 through 56.

11. Create your own policy to fetch all the roles from AWS accounts.

The screenshot shows the 'Create policy' wizard. At the top, there are three tabs: 'Visual editor' (disabled), 'JSON' (selected and highlighted with a red box), and 'Import managed policy'. Below the tabs is a large code editor containing a JSON policy document:

```

1  {
2      "Version": "2012-10-17",
3      "Statement": [
4          {
5              "Effect": "Allow",
6              "Action": [
7                  "iam>ListRoles"
8              ],
9              "Resource": "*"
10         }
11     ]
12 }

```

Below the code editor, status indicators show: Security: 0, Errors: 0, Warnings: 0, and Suggestions: 0. At the bottom right, there are 'Cancel' and 'Next: Tags' buttons, with 'Next: Tags' also highlighted with a red box.

a. In **Create policy**, select the **JSON** tab.

b. In the policy document, add the following JSON:

The screenshot shows the JSON code editor with the completed policy document:

```

JSON

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam>ListRoles"
            ],
            "Resource": "*"
        }
    ]
}

```

c. Click **Next: Tags**.

12. You can also add the required tags in the below page and click **Next: Review**.

Create policy

Add tags - optional

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

No tags associated with the resource.

Add tag

You can add up to 50 more tags.

1 2 3

13. Define the new policy.

Create policy

Review policy

Name*

Use alphanumeric and '+=_@-' characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and '+=_@-' characters.

Summary

Q Filter			
Service	Access level	Resource	Request condition
Allow (1 of 370 services) Show remaining 369			
IAM	Limited: List	All resources	None

Tags

Key	Value
No tags associated with the resource.	

* Required Cancel Previous Create policy

a. For **Name**, enter **AzureAD_SSOUserRole_Policy**.

b. For **Description**, enter **This policy will allow to fetch the roles from AWS accounts.**

c. Select **Create policy**.

14. Create a new user account in the AWS IAM service.

a. In the AWS IAM console, select **Users** and click **Add users**.

The screenshot shows the AWS Identity and Access Management (IAM) console. On the left, there's a sidebar with options like Dashboard, Access management, User groups, **Users** (which is selected and highlighted with a red box), Roles, Policies, Identity providers, and Account settings. The main area is titled 'Users' and shows a message: 'Ready to streamline human access to AWS and cloud apps?'. It says 'Identity Center is enabled. We recommend managing workforce users' access to AWS accounts and cloud applications in Identity Center.' Below this, there's a table with one row, showing 'Users (5)'. A 'Find users by username or access key' search bar is at the bottom. On the right side of the table, there are 'Add users' (highlighted with a red box), 'Delete', and other buttons.

b. In the **Specify user details** section, enter the user name as **AzureADRoleManager** and select **Next**.

The screenshot shows the 'Create user' wizard, Step 1: Specify user details. On the left, there's a sidebar with Step 1: Specify user details, Step 2: Set permissions, and Step 3: Review and create. The main area is titled 'Specify user details' and has a 'User details' section. It includes a 'User name' input field (highlighted with a red box), a note about character restrictions, a checkbox for 'Provide user access to the AWS Management Console - optional', and a callout for generating programmatic access keys. At the bottom right are 'Cancel' and 'Next' buttons, with 'Next' highlighted with a red box.

c. Create a new policy for this user.

Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. [Learn more](#)

Permissions options

- Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.
- Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1121)
Choose one or more policies to attach to your new user.

<input type="checkbox"/>	Policy name	Type	Attached entities
<input type="checkbox"/>	AccessAnalyzerService...	AWS managed	0
<input type="checkbox"/>	AdministratorAccess	AWS managed - job function	6
<input type="checkbox"/>	AdministratorAccess-A...	AWS managed	0
<input type="checkbox"/>	AdministratorAccess-A...	AWS managed	0

Permissions boundary - optional
Set a permissions boundary to control the maximum permissions for this user. Use this advanced feature used to delegate permission management to others. [Learn more](#)

Cancel Previous Next

d. Select **Attach existing policies directly**.

e. Search for the newly created policy in the filter section **AzureAD_SSOUserRole_Policy**.

f. Select the policy, and then select **Next**.

15. Review your choices and select **Create user**.

16. To download the user credentials of a user, enable the console access in **Security credentials** tab.

AzureADRoleManager

Summary

ARN arn:aws:iam::<Account_ID>:user/AzureADRoleManager	Console access Disabled	Access key 1 Not enabled
Created	Last console sign-in -	Access key 2 Not enabled

Permissions Groups Tags **Security credentials** Access Advisor

Console sign-in

Console sign-in link
https://<Account_ID>.signin.aws.amazon.com/console

Console password
Not enabled

Enable console access

17. Enter these credentials into the Microsoft Entra user provisioning section to fetch the roles from the AWS console.

Console password

 You have successfully enabled the user's new password.
This is the only time you can view this password. After you close this window, if the password is lost, you must create a new one.

Console sign-in URL

User name

Console password
[Show](#)

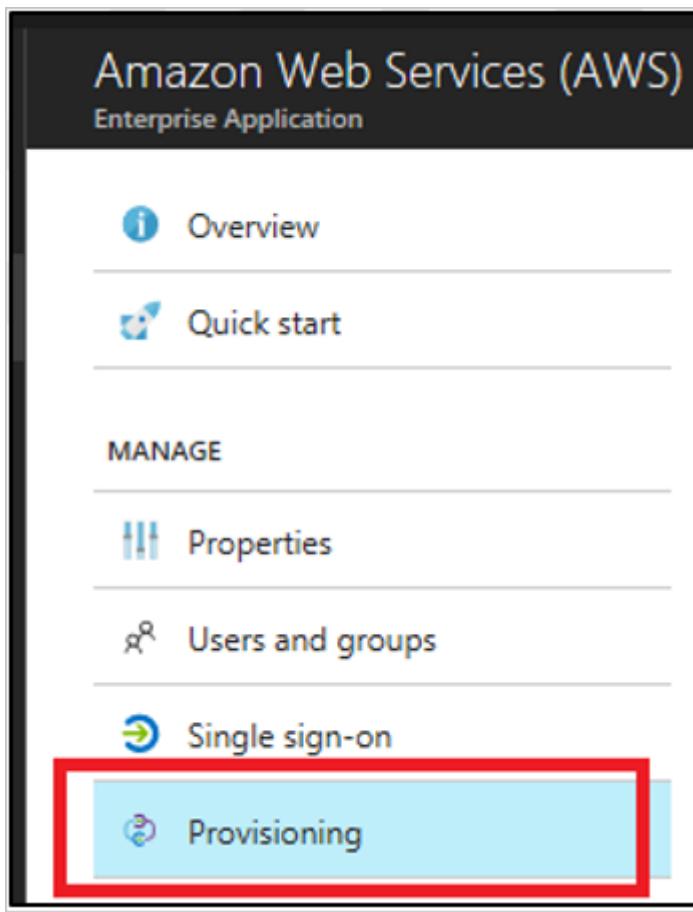
[Download .csv file](#) [Close](#)

 **Note**

AWS has a set of permissions/limits required to configure AWS SSO. To know more information on AWS limits, please refer [this](#) page.

How to configure role provisioning in AWS Single-Account Access

1. In the Microsoft Entra management portal, in the AWS app, go to **Provisioning**.



2. Enter the access key and secret in the **clientsecret** and **Secret Token** fields, respectively.

Admin Credentials

Azure AD needs the following information to connect to Amazon Web Services (AWS)'s API and synchronize user data.

* clientsecret (info) ✓

* Secret Token (info) ✓

Test Connection

- a. Enter the AWS user access key in the **clientsecret** field.
- b. Enter the AWS user secret in the **Secret Token** field.
- c. Select **Test Connection**.
- d. Save the setting by selecting **Save**.
3. In the **Settings** section, for **Provisioning Status**, select **On**. Then select **Save**.

Settings

Start and stop provisioning to Amazon Web Services (AWS), and view provisioning status.

Provisioning Status



Off

ⓘ Note

The provisioning service imports roles only from AWS to Microsoft Entra ID. The service does not provision users and groups from Microsoft Entra ID to AWS.

ⓘ Note

After you save the provisioning credentials, you must wait for the initial sync cycle to run. Sync usually takes around 40 minutes to finish. You can see the status at the bottom of the **Provisioning** page, under **Current Status**.

Create AWS Single-Account Access test user

The objective of this section is to create a user called B.Simon in AWS Single-Account Access. AWS Single-Account Access doesn't need a user to be created in their system for SSO, so you don't need to perform any action here.

Test SSO

In this section, you test your Microsoft Entra single sign-on configuration with following options.

SP initiated:

- Click on **Test this application**, this will redirect to AWS Single-Account Access Sign on URL where you can initiate the login flow.
- Go to AWS Single-Account Access Sign-on URL directly and initiate the login flow from there.

IDP initiated:

- Click on **Test this application**, and you should be automatically signed in to the AWS Single-Account Access for which you set up the SSO.

You can also use Microsoft My Apps to test the application in any mode. When you click the AWS Single-Account Access tile in the My Apps, if configured in SP mode you would be redirected to the application sign on page for initiating the login flow and if configured in IDP mode, you should be automatically signed in to the AWS Single-Account Access for which you set up the SSO. For more information about the My Apps, see [Introduction to the My Apps](#).

Known issues

- AWS Single-Account Access provisioning integration cannot be used in the AWS China regions.
- In the **Provisioning** section, the **Mappings** subsection shows a "Loading..." message, and never displays the attribute mappings. The only provisioning workflow supported today is the import of roles from AWS into Microsoft Entra ID for selection during a user or group assignment. The attribute mappings for this are predetermined, and aren't configurable.
- The **Provisioning** section only supports entering one set of credentials for one AWS tenant at a time. All imported roles are written to the `appRoles` property of the Microsoft Entra ID [servicePrincipal object](#) for the AWS tenant.

Multiple AWS tenants (represented by `servicePrincipals`) can be added to Microsoft Entra ID from the gallery for provisioning. There's a known issue, however, with not being able to automatically write all of the imported roles from the multiple AWS `servicePrincipals` used for provisioning into the single `servicePrincipal` used for SSO.

As a workaround, you can use the [Microsoft Graph API](#) to extract all of the `appRoles` imported into each AWS `servicePrincipal` where provisioning is configured. You can subsequently add these role strings to the AWS `servicePrincipal` where SSO is configured.

- Roles must meet the following requirements to be eligible to be imported from AWS into Microsoft Entra ID:
 - Roles must have exactly one saml-provider defined in AWS
 - The combined length of the ARN(Amazon Resource Name) for the role and the ARN for the associated saml-provider must be less than 240 characters.

Change log

- 01/12/2020 - Increased role length limit from 119 characters to 239 characters.

Next steps

Once you configure AWS Single-Account Access you can enforce Session Control, which protects exfiltration and infiltration of your organization's sensitive data in real time.

Session Control extends from Conditional Access. [Learn how to enforce session control with Microsoft Defender for Cloud Apps.](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

Tutorial: Microsoft Entra SSO integration with AWS IAM Identity Center

Article • 01/19/2024

In this tutorial, you'll learn how to integrate AWS IAM Identity Center (successor to AWS Single Sign-On) with Microsoft Entra ID. When you integrate AWS IAM Identity Center with Microsoft Entra ID, you can:

- Control in Microsoft Entra ID who has access to AWS IAM Identity Center.
- Enable your users to be automatically signed-in to AWS IAM Identity Center with their Microsoft Entra accounts.
- Manage your accounts in one central location.

Prerequisites

To get started, you need the following items:

- A Microsoft Entra subscription. If you don't have a subscription, you can get a [free account](#).
- AWS IAM Identity Center enabled subscription.

Scenario description

In this tutorial, you configure and test Microsoft Entra SSO in a test environment.

- AWS IAM Identity Center supports **SP and IDP initiated SSO**.
- AWS IAM Identity Center supports [Automated user provisioning](#).

Add AWS IAM Identity Center from the gallery

To configure the integration of AWS IAM Identity Center into Microsoft Entra ID, you need to add AWS IAM Identity Center from the gallery to your list of managed SaaS apps.

1. Sign in to the [Microsoft Entra admin center](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > New application**.
3. In the **Add from the gallery** section, type **AWS IAM Identity Center** in the search box.

4. Select **AWS IAM Identity Center** from results panel and then add the app. Wait a few seconds while the app is added to your tenant.

Alternatively, you can also use the [Enterprise App Configuration Wizard](#). In this wizard, you can add an application to your tenant, add users/groups to the app, assign roles, as well as walk through the SSO configuration as well. [Learn more about Microsoft 365 wizards](#).

Configure and test Microsoft Entra SSO for AWS IAM Identity Center

Configure and test Microsoft Entra SSO with AWS IAM Identity Center using a test user called **B.Simon**. For SSO to work, you need to establish a link relationship between a Microsoft Entra user and the related user in AWS IAM Identity Center.

To configure and test Microsoft Entra SSO with AWS IAM Identity Center, perform the following steps:

1. [Configure Microsoft Entra SSO](#) - to enable your users to use this feature.
 - a. [Create a Microsoft Entra test user](#) - to test Microsoft Entra single sign-on with B.Simon.
 - b. [Assign the Microsoft Entra test user](#) - to enable B.Simon to use Microsoft Entra single sign-on.
2. [Configure AWS IAM Identity Center SSO](#) - to configure the single sign-on settings on application side.
 - a. [Create AWS IAM Identity Center test user](#) - to have a counterpart of B.Simon in AWS IAM Identity Center that is linked to the Microsoft Entra representation of user.
3. [Test SSO](#) - to verify whether the configuration works.

Configure Microsoft Entra SSO

Follow these steps to enable Microsoft Entra SSO.

1. Sign in to the [Microsoft Entra admin center](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > AWS IAM Identity Center > Single sign-on**.
3. On the **Select a single sign-on method** page, select **SAML**.

4. On the **Set up single sign-on with SAML** page, click the pencil icon for **Basic SAML Configuration** to edit the settings.

The screenshot shows the 'Set up Single Sign-On with SAML' page. At the top, there's a brief introduction about SSO implementation. Below it, a note says 'Read the configuration guide' with a link. The main area has a numbered step '1' next to 'Basic SAML Configuration'. This section contains fields for 'Identifier (Entity ID)', 'Reply URL (Assertion Consumer Service URL)', 'Sign on URL', 'Relay State (Optional)', and 'Logout Url (Optional)'. To the right of these fields is a red-bordered 'Edit' button with a pencil icon.

5. If you have **Service Provider metadata file**, on the **Basic SAML Configuration** section, perform the following steps:

- Click **Upload metadata file**.
- Click on **folder logo** to select metadata file which is explained to download in **Configure AWS IAM Identity Center SSO** section and click **Add**.

The screenshot shows a dialog box for uploading a metadata file. It includes buttons for 'Upload metadata file', 'Change single sign-on mode', 'Test this application', and 'Got feedback?'. Below these is a section titled 'Upload metadata file.' with a note about values being provided by the application. There's a file selection input field with a 'Select a file' placeholder and a blue 'Add' button. A 'Cancel' button is also present.

- Once the metadata file is successfully uploaded, the **Identifier** and **Reply URL** values get auto populated in Basic SAML Configuration section.

! Note

If the **Identifier** and **Reply URL** values are not getting auto populated, then fill in the values manually according to your requirement.

! Note

When changing identity provider in AWS (i.e. from AD to external provider such as Microsoft Entra ID) the AWS metadata will change and need to be reuploaded to Azure for SSO to function correctly.

6. If you don't have **Service Provider metadata file**, perform the following steps on the **Basic SAML Configuration** section, if you wish to configure the application in IDP initiated mode, perform the following steps:

a. In the **Identifier** text box, type a URL using the following pattern:

`https://<REGION>.signin.aws.amazon.com/platform/saml/<ID>`

b. In the **Reply URL** text box, type a URL using the following pattern:

`https://<REGION>.signin.aws.amazon.com/platform/saml/acs/<ID>`

7. Click **Set additional URLs** and perform the following step if you wish to configure the application in **SP** initiated mode:

In the **Sign-on URL** text box, type a URL using the following pattern:

`https://portal.sso.<REGION>.amazonaws.com/saml/assertion/<ID>`

① Note

These values are not real. Update these values with the actual Identifier, Reply URL and Sign-on URL. Contact **AWS IAM Identity Center Client support team** to get these values. You can also refer to the patterns shown in the **Basic SAML Configuration** section.

8. AWS IAM Identity Center application expects the SAML assertions in a specific format, which requires you to add custom attribute mappings to your SAML token attributes configuration. The following screenshot shows the list of default attributes.

Attributes & Claims	
givenname	user.givenname
surname	user.surname
emailaddress	user.mail
name	user.userprincipalname
Unique User Identifier	user.userprincipalname

① Note

If ABAC is enabled in AWS IAM Identity Center, the additional attributes may be passed as session tags directly into AWS accounts.

9. On the **Set-up single sign-on with SAML** page, in the **SAML Signing Certificate** section, find **Federation Metadata XML** and select **Download** to download the

certificate and save it on your computer.

3 SAML Signing Certificate

Status Active

Thumbprint <Thumbprintvalue>

Expiration <Expiration>

Notification Email <Email address>

App Federation Metadata Url <App Federation Metadata Url>

Certificate (Base64) Download

Certificate (Raw) Download

Federation Metadata XML Download

10. On the **Set up AWS IAM Identity Center** section, copy the appropriate URL(s) based on your requirement.

Set up <Application Name>

You'll need to configure the application to link with Microsoft Entra ID.

Login URL https://login.microsoftonline.com/4f74... ↗

Microsoft Entra ID Identifier https://sts.windows.net/4f7437a6-3d76... ↗

Logout URL https://login.microsoftonline.com/4f74... ↗

Create a Microsoft Entra test user

In this section, you'll create a test user called B.Simon.

1. Sign in to the [Microsoft Entra admin center](#) as at least a **User Administrator**.
2. Browse to **Identity** > **Users** > **All users**.
3. Select **New user** > **Create new user**, at the top of the screen.
4. In the **User properties**, follow these steps:
 - a. In the **Display name** field, enter **B.Simon**.
 - b. In the **User principal name** field, enter the **username@companydomain.extension**. For example, **B.Simon@contoso.com**.
 - c. Select the **Show password** check box, and then write down the value that's displayed in the **Password** box.
 - d. Select **Review + create**.
5. Select **Create**.

Assign the Microsoft Entra test user

In this section, you'll enable B.Simon to use single sign-on by granting access to AWS IAM Identity Center.

1. Sign in to the [Microsoft Entra admin center](#) as at least a **Cloud Application Administrator**.
2. Browse to **Identity > Applications > Enterprise applications > AWS IAM Identity Center**.
3. In the app's overview page, select **Users and groups**.
4. Select **Add user/group**, then select **Users and groups** in the **Add Assignment** dialog.
 - a. In the **Users and groups** dialog, select **B.Simon** from the Users list, then click the **Select** button at the bottom of the screen.
 - b. If you are expecting a role to be assigned to the users, you can select it from the **Select a role** dropdown. If no role has been set up for this app, you see "Default Access" role selected.
 - c. In the **Add Assignment** dialog, click the **Assign** button.

Configure AWS IAM Identity Center SSO

1. In a different web browser window, sign in to your AWS IAM Identity Center company site as an administrator
2. Go to the **Services -> Security, Identity, & Compliance -> AWS IAM Identity Center**.
3. In the left navigation pane, choose **Settings**.
4. On the **Settings** page, find **Identity source**, click on **Actions** pull-down menu, and select **Change identity source**.

The screenshot shows the IAM Identity Center Settings page. The left sidebar has a 'Settings' section highlighted with a red box. The main area shows 'Details' for managing identity sources and multi-factor authentication. It lists 'Region' as 'US East (N. Virginia) | us-east-1'. Under 'Identity source', there are fields for 'Identity source' (set to 'Identity Center directory'), 'Authentication method' (set to 'Password'), 'Provisioning method' (set to 'Direct'), and 'AWS access portal URL' (set to '<https://access-portal>'). A red box highlights the 'Actions' button and the 'Change identity source' link.

5. On the Change identity source page, choose **External identity provider**.

The screenshot shows the 'Change identity source' wizard. Step 1: Choose identity source. It shows three options: 'Identity Center directory', 'Active Directory', and 'External identity provider'. The 'External identity provider' option is selected and highlighted with a red box. The 'Next' button is also highlighted with a red box.

6. Perform the below steps in the **Configure external identity provider** section:

Change identity source

Your identity source is where you manage users and groups. You use IAM Identity Center to manage permissions for users and groups in your identity source to access AWS accounts and cloud applications. [Learn more](#)

Step 1
Choose identity source

Step 2
Configure external identity provider

Step 3
Confirm change

Configure external identity provider

Service provider metadata

Your identity provider (IdP) requires the following IAM Identity Center certificate and metadata information to trust IAM Identity Center as a service provider. You can copy and paste this information, type it in the service provider configuration interface for your IdP, or download the IAM Identity Center metadata file and upload it to your IdP.

[Download metadata file](#)



AWS access portal sign-in URL

[<sign-in URL>](#)

IAM Identity Center Assertion Consumer Service (ACS) URL

[<ACS URL>](#)

IAM Identity Center issuer URL

[<issuer URL>](#)

Identity provider metadata

AWS requires specific metadata provided by your identity provider (IdP) to establish trust. You may copy and paste from your IdP, type the metadata in manually, or upload a metadata exchange file that you download from your IdP.

Depending on the identity provider, you may have to:

IdP SAML metadata

[Choose file](#)

Or

IdP sign-in URL

IdP issuer URL

IdP certificate

[Choose file](#)

[Cancel](#)

[Previous](#)

[Next](#)

- a. In the **Service provider metadata** section, find **AWS SSO SAML metadata**, select **Download metadata file** to download the metadata file and save it on your computer and use this metadata file to upload on Azure portal.
 - b. Copy **AWS access portal sign-in URL** value, paste this value into the **Sign on URL** text box in the **Basic SAML Configuration** section.
 - c. In the **Identity provider metadata** section, select **Choose file** to upload the metadata file which you have downloaded.
 - d. Choose **Next: Review**.
7. In the text box, type **ACCEPT** to change the identity source.

Change identity source

Your identity source is where you manage users and groups. You use IAM Identity Center to manage permissions for users and groups in your identity source to access AWS accounts and cloud applications. [Learn more](#)

Step 1
Choose identity source

Step 2
Configure external identity provider

Step 3
Confirm change

Confirm change

Step 1: Choose identity source

Identity source

Identity source
External Identity provider

Step 2: Configure external identity provider

Service provider metadata

IdP SAML metadata
 .saml_metadata.xml
Size: 680 bytes
Last modified:

Review and confirm

⚠ Review the following consequences of your requested identity source change:

- You are changing your identity source to use an external identity provider (IdP).
- IAM Identity Center will delete your current multi-factor authentication (MFA) configuration.
- All current permission sets and SAML 2.0 application configurations will be retained.
- IAM Identity Center preserves your current users and groups, and their assignments. However, only users who have usernames that match the usernames in your identity provider (IdP) can authenticate.
- You must complete your identity provider (IdP) SAML configuration for IAM Identity Center so that your users can sign in. Identity Center will use your IdP for all authentications.
- You must manage your multi-factor authentication (MFA) configuration and policies in your identity provider (IdP).
- You must add (provision) all users in your identity provider (IdP) who will use IAM Identity Center before they can sign in. If you enable Systems for Cross-domain Identity Management (SCIM) to provision users and groups (recommended), your IdP will be the authoritative source of users and groups, and you must add and modify them in your IdP. Without SCIM, you can provision users and manage groups in IAM Identity Center only; all provisioned usernames must match the corresponding usernames in your IdP.
- IAM Identity Center will keep your current configuration of attributes for access control. We recommend that you review your configuration and update it after you complete the identity source change.

Confirm that you want to change your identity source by entering ACCEPT in the field below.

ACCEPT

Cancel

Previous

Change identity source

8. Click **Change identity source**.

Create AWS IAM Identity Center test user

1. Open the **AWS IAM Identity Center console**.

2. In the left navigation pane, choose **Users**.

3. On the Users page, choose **Add user**.

4. On the Add user page, follow these steps:

- a. In the **Username** field, enter B.Simon.
- b. In the **Email address** field, enter the `username@companydomain.extension`. For example, `B.Simon@contoso.com`.
- c. In the **Confirm email address** field, re-enter the email address from the previous step.
- d. In the **First name** field, enter `Britta`.
- e. In the **Last name** field, enter `Simon`.
- f. In the **Display name** field, enter `B.Simon`.
- g. Choose **Next**, and then **Next** again.

 **Note**

Make sure the username and email address entered in AWS IAM Identity Center matches the user's Microsoft Entra sign-in name. This will help avoid any authentication problems.

5. Choose **Add user**.
6. Next, you will assign the user to your AWS account. To do so, in the left navigation pane of the AWS IAM Identity Center console, choose **AWS accounts**.
7. On the AWS Accounts page, select the AWS organization tab, check the box next to the AWS account you want to assign to the user. Then choose **Assign users**.
8. On the Assign Users page, find and check the box next to the user B.Simon. Then choose **Next: Permission sets**.
9. Under the select permission sets section, check the box next to the permission set you want to assign to the user B.Simon. If you don't have an existing permission set, choose **Create new permission set**.

 **Note**

Permission sets define the level of access that users and groups have to an AWS account. To learn more about permission sets, see the [AWS IAM Identity Center Multi Account Permissions](#) page.

10. Choose **Finish**.

Note

AWS IAM Identity Center also supports automatic user provisioning, you can find more details [here](#) on how to configure automatic user provisioning.

Test SSO

In this section, you test your Microsoft Entra single sign-on configuration with following options.

SP initiated:

- Click on **Test this application**, this will redirect to AWS IAM Identity Center sign-in URL where you can initiate the login flow.
- Go to AWS IAM Identity Center sign-in URL directly and initiate the login flow from there.

IDP initiated:

- Click on **Test this application**, and you should be automatically signed in to the AWS IAM Identity Center for which you set up the SSO.

You can also use Microsoft My Apps to test the application in any mode. When you click the AWS IAM Identity Center tile in the My Apps, if configured in SP mode you would be redirected to the application sign on page for initiating the login flow and if configured in IDP mode, you should be automatically signed in to the AWS IAM Identity Center for which you set up the SSO. For more information about the My Apps, see [Introduction to the My Apps](#).

Next steps

Once you configure AWS IAM Identity Center you can enforce session control, which protects exfiltration and infiltration of your organization's sensitive data in real time. Session control extends from Conditional Access. [Learn how to enforce session control with Microsoft Defender for Cloud Apps](#).

Tutorial: Configure AWS IAM Identity Center for automatic user provisioning

Article • 11/29/2023

This tutorial describes the steps you need to perform in both AWS IAM Identity Center(successor to AWS single sign-On) and Microsoft Entra ID to configure automatic user provisioning. When configured, Microsoft Entra ID automatically provisions and de-provisions users and groups to [AWS IAM Identity Center](#) using the Microsoft Entra provisioning service. For important details on what this service does, how it works, and frequently asked questions, see [Automate user provisioning and deprovisioning to SaaS applications with Microsoft Entra ID](#).

Capabilities Supported

- ✓ Create users in AWS IAM Identity Center
- ✓ Remove users in AWS IAM Identity Center when they no longer require access
- ✓ Keep user attributes synchronized between Microsoft Entra ID and AWS IAM Identity Center
- ✓ Provision groups and group memberships in AWS IAM Identity Center
- ✓ [IAM Identity Center](#) to AWS IAM Identity Center

Prerequisites

The scenario outlined in this tutorial assumes that you already have the following prerequisites:

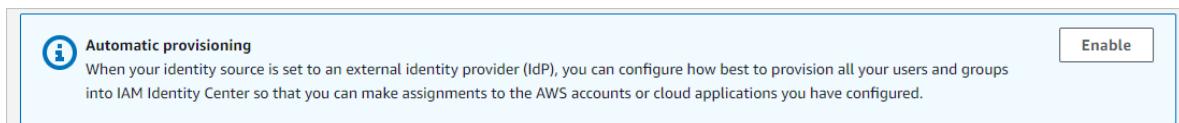
- A [Microsoft Entra tenant](#)
- A user account in Microsoft Entra ID with [permission](#) to configure provisioning (for example, Application Administrator, Cloud Application administrator, Application Owner, or Global Administrator).
- A SAML connection from your Microsoft Entra account to AWS IAM Identity Center, as described in Tutorial

Step 1: Plan your provisioning deployment

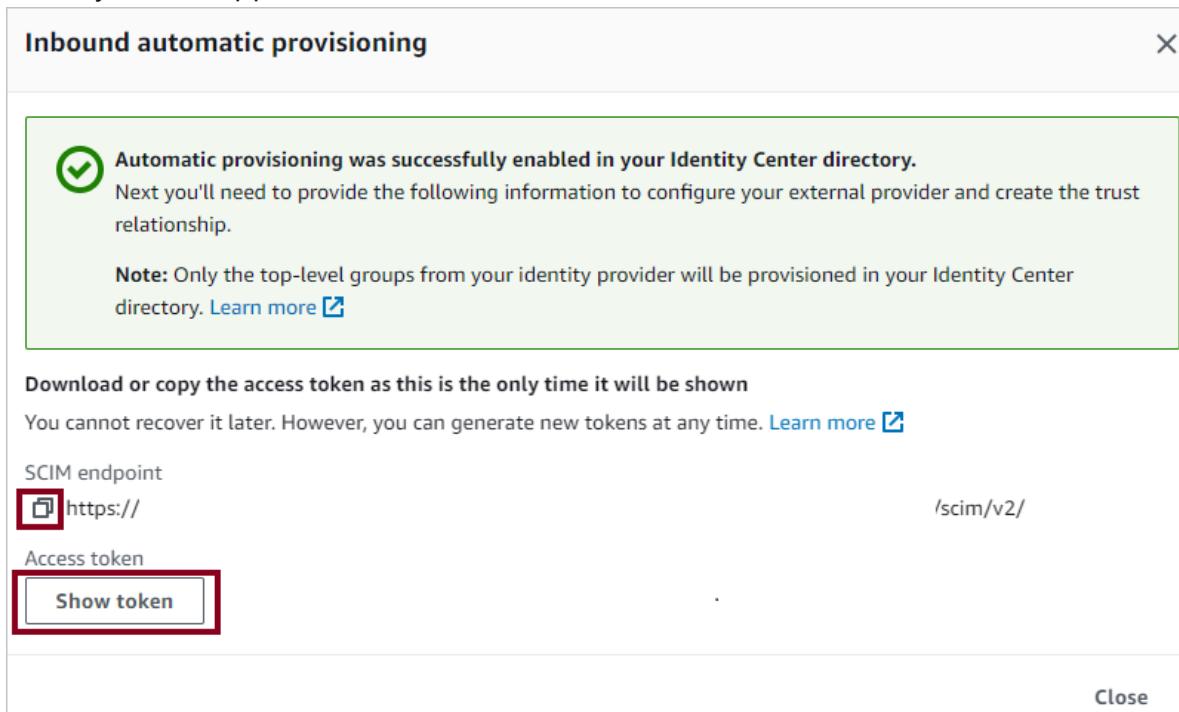
1. Learn about [how the provisioning service works](#).
2. Determine who is in [scope for provisioning](#).
3. Determine what data to [map between Microsoft Entra ID and AWS IAM Identity Center](#).

Step 2: Configure AWS IAM Identity Center to support provisioning with Microsoft Entra ID

1. Open the [AWS IAM Identity Center](#).
2. Choose **Settings** in the left navigation pane
3. In **Settings**, click on **Enable** in the Automatic provisioning section.



4. In the Inbound automatic provisioning dialog box, copy and save the **SCIM endpoint** and **Access Token** (visible after clicking on Show Token). These values are entered in the **Tenant URL** and **Secret Token** field in the Provisioning tab of your AWS IAM Identity Center application.



Step 3: Add AWS IAM Identity Center from the Microsoft Entra application gallery

Add AWS IAM Identity Center from the Microsoft Entra application gallery to start managing provisioning to AWS IAM Identity Center. If you have previously setup AWS IAM Identity Center for SSO, you can use the same application. Learn more about adding an application from the gallery [here](#).

Step 4: Define who is in scope for provisioning

The Microsoft Entra provisioning service allows you to scope who is provisioned based on assignment to the application and or based on attributes of the user / group. If you choose to scope who is provisioned to your app based on assignment, you can use the following [steps](#) to assign users and groups to the application. If you choose to scope who is provisioned based solely on attributes of the user or group, you can use a scoping filter as described [here](#).

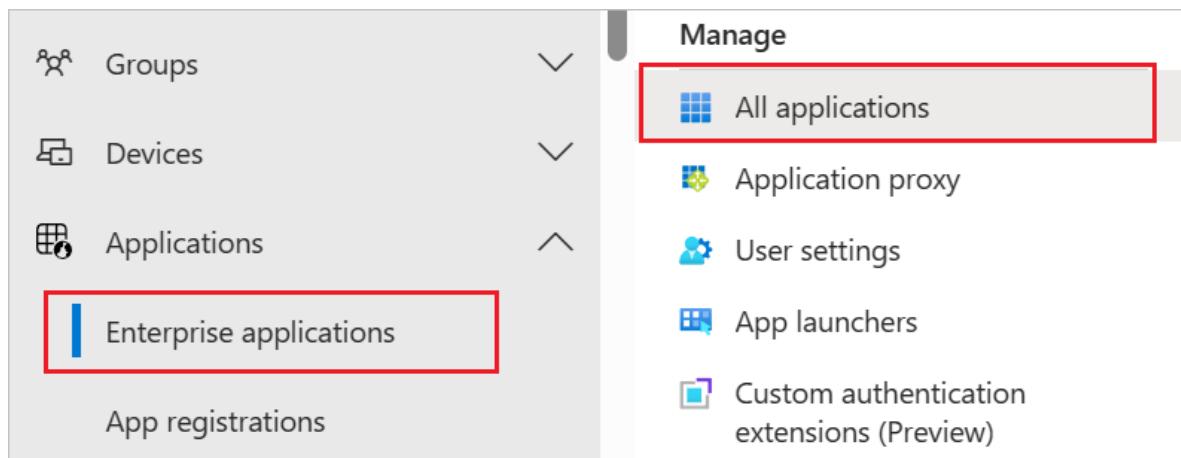
- Start small. Test with a small set of users and groups before rolling out to everyone. When scope for provisioning is set to assigned users and groups, you can control this by assigning one or two users or groups to the app. When scope is set to all users and groups, you can specify an [attribute based scoping filter](#).
- If you need additional roles, you can [update the application manifest](#) to add new roles.

Step 5: Configure automatic user provisioning to AWS IAM Identity Center

This section guides you through the steps to configure the Microsoft Entra provisioning service to create, update, and disable users and/or groups in TestApp based on user and/or group assignments in Microsoft Entra ID.

To configure automatic user provisioning for AWS IAM Identity Center in Microsoft Entra ID:

1. Sign in to the [Microsoft Entra admin center](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications**



3. In the applications list, select AWS IAM Identity Center.

[+ New application](#) [Refresh](#) [Download \(Export\)](#) | [Preview info](#) | [Columns](#) | ..

View, filter, and search applications in your organization that are set up to use your Microsoft Entra tenant as their Identity Provider.

The list of applications that are maintained by your organization are in [application registrations](#).

Search by application name or object ID

Application type == **Enterprise Applications** [X](#) Application ID starts with [X](#)

[+ Add filters](#)

4. Select the **Provisioning** tab.

Manage

[Properties](#)

[Owners](#)

[Users and groups](#)

[Single sign-on](#)

Provisioning

[Self-service](#)

5. Set the **Provisioning Mode** to **Automatic**.

Provisioning Mode **Manual**

Use the tools and ac
and de-provision the user account records stored in Microsoft Cloud - [Configure](#)

Automatic

6. Under the **Admin Credentials** section, input your AWS IAM Identity Center **Tenant URL** and **Secret Token** retrieved earlier in Step 2. Click **Test Connection** to ensure Microsoft Entra ID can connect to AWS IAM Identity Center.

Tenant URL * [?](#)

Secret Token

Test Connection

7. In the **Notification Email** field, enter the email address of a person or group who should receive the provisioning error notifications and select the **Send an email notification when a failure occurs** check box.

Notification Email 	<input type="text"/>
<input type="checkbox"/> Send an email notification when a failure occurs	

8. Select Save.

9. Under the **Mappings** section, select **Synchronize Microsoft Entra users to AWS IAM Identity Center**.

10. Review the user attributes that are synchronized from Microsoft Entra ID to AWS IAM Identity Center in the **Attribute-Mapping** section. The attributes selected as **Matching** properties are used to match the user accounts in AWS IAM Identity Center for update operations. If you choose to change the [matching target attribute](#), you need to ensure that the AWS IAM Identity Center API supports filtering users based on that attribute. Select the **Save** button to commit any changes.

Attribute	Type	Supported for Filtering
userName	String	✓
active	Boolean	
displayName	String	
title	String	
emails[type eq "work"].value	String	
preferredLanguage	String	
name.givenName	String	
name.familyName	String	
name.formatted	String	
addresses[type eq "work"].formatted	String	
addresses[type eq "work"].streetAddress	String	
addresses[type eq "work"].locality	String	
addresses[type eq "work"].region	String	
addresses[type eq "work"].postalCode	String	
addresses[type eq "work"].country	String	
phoneNumbers[type eq "work"].value	String	

Attribute	Type	Supported for Filtering
externalId	String	
locale	String	
timezone	String	
urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:employeeNumber	String	
urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:department	String	
urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:division	String	
urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:costCenter	String	
urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:organization	String	
urn:ietf:params:scim:schemas:extension:enterprise:2.0:User:manager	Reference	

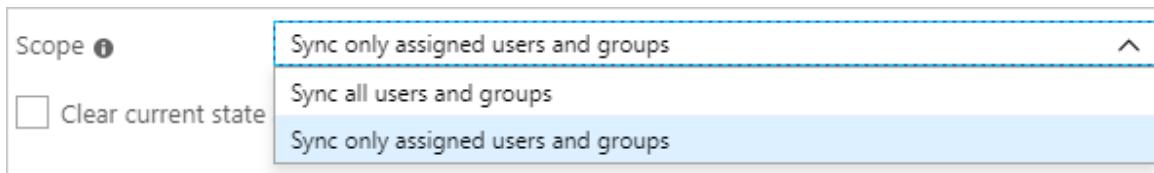
11. Under the **Mappings** section, select **Synchronize Microsoft Entra groups to AWS IAM Identity Center**.
12. Review the group attributes that are synchronized from Microsoft Entra ID to AWS IAM Identity Center in the **Attribute-Mapping** section. The attributes selected as **Matching** properties are used to match the groups in AWS IAM Identity Center for update operations. Select the **Save** button to commit any changes.

Attribute	Type	Supported for Filtering
displayName	String	✓
externalId	String	
members	Reference	

13. To configure scoping filters, refer to the following instructions provided in the [Scoping filter tutorial](#).
14. To enable the Microsoft Entra provisioning service for AWS IAM Identity Center, change the **Provisioning Status** to **On** in the **Settings** section.



15. Define the users and/or groups that you would like to provision to AWS IAM Identity Center by choosing the desired values in **Scope** in the **Settings** section.



16. When you're ready to provision, click **Save**.



This operation starts the initial synchronization cycle of all users and groups defined in **Scope** in the **Settings** section. The initial cycle takes longer to perform than subsequent cycles, which occur approximately every 40 minutes as long as the Microsoft Entra provisioning service is running.

Step 6: Monitor your deployment

Once you configure provisioning, use the following resources to monitor your deployment:

1. Use the [provisioning logs](#) to determine which users were provisioned successfully or unsuccessfully
2. Check the [progress bar](#) to see the status of the provisioning cycle and how close it is to completion
3. If the provisioning configuration seems to be in an unhealthy state, the application goes into quarantine. Learn more about quarantine states [here](#).

Just-in-time (JIT) application access with PIM for groups (preview)

With PIM for Groups, you can provide just-in-time access to groups in Amazon Web Services and reduce the number of users that have permanent access to privileged groups in AWS.

Configure your enterprise application for SSO and provisioning

1. Add AWS IAM Identity Center to your tenant, configure it for provisioning as described in the tutorial above, and start provisioning.
2. Configure [single sign-on](#) for AWS IAM Identity Center.
3. Create a [group](#) that will provide all users access to the application.
4. Assign the group to the AWS Identity Center application.
5. Assign your test user as a direct member of the group created in the previous step, or provide them access to the group through an access package. This group can be used for persistent, non-admin access in AWS.

Enable PIM for groups

1. Create a second group in Microsoft Entra ID. This group will provide access to admin permissions in AWS.
2. Bring the group under [management in Microsoft Entra PIM](#).
3. Assign your test user as [eligible for the group in PIM](#) with the role set to member.
4. Assign the second group to the AWS IAM Identity Center application.
5. Use on-demand provisioning to create the group in AWS IAM Identity Center.
6. Sign-in to AWS IAM Identity Center and assign the second group the necessary permissions to perform admin tasks.

Now any end user that was made eligible for the group in PIM can get JIT access to the group in AWS by [activating their group membership](#).

Key considerations

- The group membership is generally updated within 2 - 10 minutes of requesting access to the group. Please wait before attempting to sign-in to AWS. If the user is unable to access the necessary group in AWS, please review the troubleshooting tips below, PIM logs, and provisioning logs to ensure that the group membership was updated successfully. Depending on how the target application has been architected, it may take additional time for the group membership to take effect in the application.
- Deactivation is done during the regular incremental cycle. It isn't processed immediately through on-demand provisioning.
- The just-in-time integration between PIM and on-demand provisioning supports 5 activation requests every 10 seconds per AWS application.

<https://www.youtube-nocookie.com/embed/aXp2CUFe7vk>

Troubleshooting Tips

Missing attributes

When provisioning a user to AWS, they're required to have the following attributes

- firstName
- lastName
- displayName
- userName

Users who don't have these attributes will fail with the following error

```
ERROR: StatusCode: BadRequest  
Message: Processing of the HTTP request resulted in an exception. Please see the HTTP response returned by the 'Response' property of this exception for details.  
Web Response:  
{"schema":["urn:ietf:params:scim:api:messages:2.0:Error"],"detail":"Request is unparsable, syntactically incorrect, or violates schema.","status":"400",
```

Multi-valued attributes

AWS doesn't support the following multi-valued attributes:

- email
- phone numbers

Trying to flow the above as multi-valued attributes will result in the following error message

```
ERROR: StatusCode: BadRequest  
Message: Processing of the HTTP request resulted in an exception. Please see the HTTP response returned by the 'Response' property of this exception for details.  
Web Response:  
{"schema":["urn:ietf:params:scim:api:messages:2.0:Error"],"detail":"Request is unparsable, syntactically incorrect, or violates schema.","status":"400",
```

There are two ways to resolve this

1. Ensure the user only has one value for phoneNumber/email
2. Remove the duplicate attributes. For example, having two different attributes being mapped from Microsoft Entra ID both mapped to "phoneNumber__" on the AWS side would result in the error if both attributes have values in Microsoft Entra ID. Only having one attribute mapped to a "phoneNumber__" attribute would resolve the error.

Invalid characters

Currently AWS IAM Identity Center isn't allowing some other characters that Microsoft Entra ID supports like tab (\t), new line (\n), return carriage (\r), and characters such as "<|>|;|:%".

You can also check the AWS IAM Identity Center troubleshooting tips [here](#) for more troubleshooting tips

Additional resources

- [Managing user account provisioning for Enterprise Apps](#)
- [What is application access and IAM Identity Center with Microsoft Entra ID?](#)

Next steps

- [Learn how to review logs and get reports on provisioning activity](#)

Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#) | [Get help at Microsoft Q&A](#)

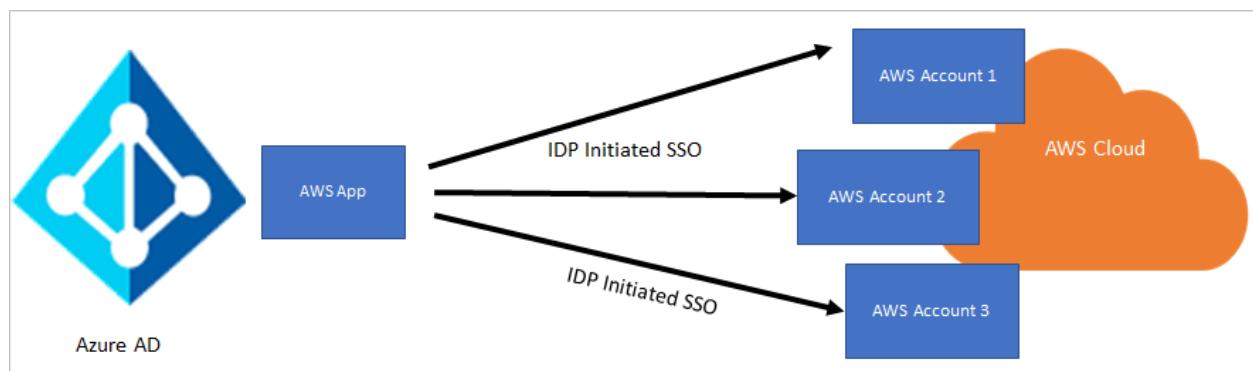
Tutorial: Microsoft Entra integration with Amazon Web Services

Article • 10/23/2023

In this tutorial, you learn how to integrate Microsoft Entra ID with Amazon Web Services (AWS) (legacy tutorial).

This integration provides the following benefits:

- You can control in Microsoft Entra ID who has access to AWS.
- You can enable your users to automatically sign in to AWS by using single sign-on (SSO) with their Microsoft Entra accounts.
- You can manage your accounts in one central location, the Azure portal.



ⓘ Note

We recommend that you *not* connect one AWS app to all your AWS accounts. Instead, we recommend that you use [Microsoft Entra SSO integration with AWS](#) to configure multiple instances of your AWS account to multiple instances of AWS apps in Microsoft Entra ID.

We recommend that you *not* connect one AWS app to all your AWS accounts, for the following reasons:

- Use this approach only if you have a small number of AWS accounts and roles, because this model isn't scalable as the number of AWS accounts and the roles within them increase. The approach doesn't use AWS role-import functionality with Microsoft Entra user provisioning, so you have to manually add, update, or delete the roles.
- You have to use the Microsoft Graph Explorer approach to patch all the roles to the app. We don't recommend using the manifest file approach.

- Customers report that after they've added ~1,200 app roles for a single AWS app, any further operation on the app starts throwing the errors related to size. There is a hard size limit to the application object.
- You have to manually update the roles as they get added in any of the accounts. This is unfortunately a *replace* approach, not an *append* approach. Also, if your account numbers are growing, this becomes an $n \times n$ relationship with accounts and roles.
- All the AWS accounts use the same federation metadata XML file. At the time of certificate rollover, updating the certificate on all the AWS accounts at the same time can be a massive exercise.

Prerequisites

To configure Microsoft Entra integration with AWS, you need the following items:

- A Microsoft Entra subscription. If you don't have a Microsoft Entra subscription, you can get a [one-month trial ↗](#).
- An AWS SSO-enabled subscription.

ⓘ Note

We do not recommend that you test the steps in this tutorial in a production environment unless it is necessary.

Scenario description

In this tutorial, you configure and test Microsoft Entra SSO in a test environment.

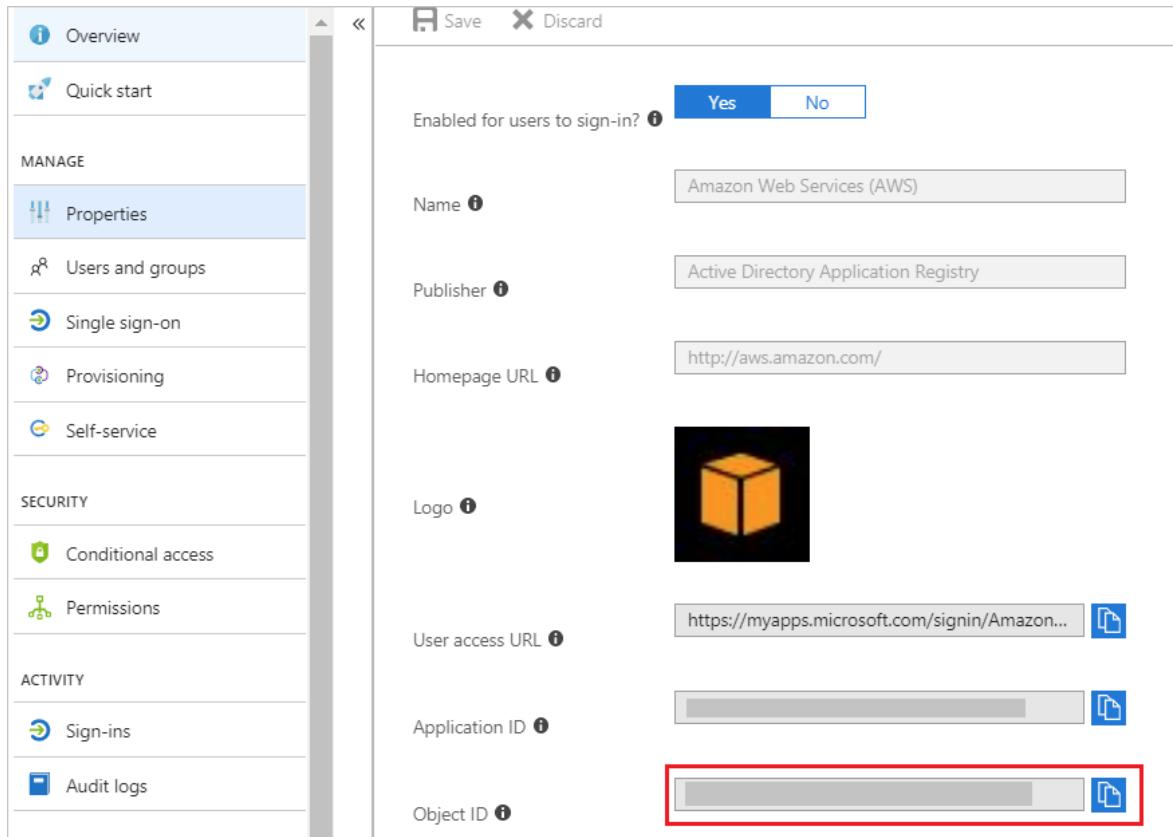
AWS supports SP-initiated and IDP-initiated SSO.

Add AWS from the gallery

To configure the integration of AWS into Microsoft Entra ID, you add AWS from the gallery to your list of managed software as a service (SaaS) apps.

1. Sign in to the [Microsoft Entra admin center ↗](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > New application**.

3. In the Add from the gallery section, type **Amazon Web Services** in the search box.
4. In the results list, select **Amazon Web Services**, and then add the app. In a few seconds, the app is added to your tenant.
5. Go to the **Properties** pane, and then copy the value that's displayed in the **Object ID** box.



Configure and test Microsoft Entra SSO

In this section, you configure and test Microsoft Entra single sign-on with AWS based on a test user called "Britta Simon."

For single sign-on to work, Microsoft Entra ID needs to know what the counterpart user in AWS is to the Microsoft Entra user. In other words, a link relationship between the Microsoft Entra user and the same user in AWS needs to be established.

In AWS, assign the value of the **user name** in Microsoft Entra ID as the value of the **AWS Username** to establish the link relationship.

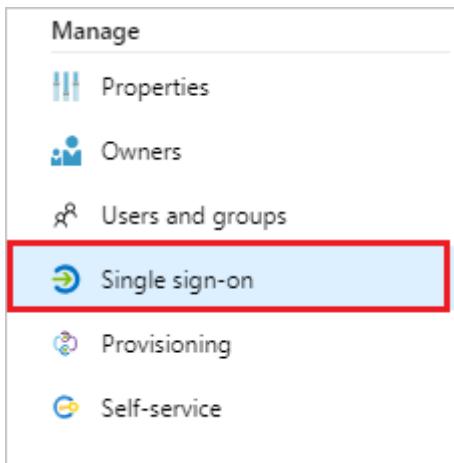
To configure and test Microsoft Entra single sign-on with AWS, do the following:

1. [Configure Microsoft Entra SSO](#) to enable your users to use this feature.
2. [Configure AWS SSO](#) to configure SSO settings on the application side.
3. [Test SSO](#) to verify that the configuration works.

Configure Microsoft Entra SSO

In this section, you enable Microsoft Entra SSO in the Azure portal and configure SSO in your AWS application by doing the following:

1. Sign in to the [Microsoft Entra admin center](#) as at least a [Cloud Application Administrator](#).
2. Browse to **Identity > Applications > Enterprise applications > Amazon Web Services (AWS)**.
3. select **Single sign-on**.



4. On the **Select a single sign-on method** pane, select **SAML/WS-Fed** mode to enable single sign-on.

The screenshot shows a pane titled "Select a single sign-on method" with a "Help me decide" link. There are three options displayed:

- Disabled**: An icon of a circle with a slash. Description: "User must manually enter their username and password."
- SAML**: An icon of a gear. Description: "Rich and secure authentication to applications using the SAML (Security Assertion Markup Language) protocol."
- Linked**: An icon of two overlapping circles. Description: "Link to an application in My Apps and/or Office 365 application launcher."

5. On the **Set up Single Sign-On with SAML** pane, select the **Edit** button (pencil icon).

Set up Single Sign-On with SAML

An SSO implementation based on federation protocols improves security, reliability, and end user experiences and is easier to implement. Choose SAML single sign-on whenever possible for existing applications that do not use OpenID Connect or OAuth. [Learn more.](#)

Read the [configuration guide](#) for help integrating <App Name>

1

Basic SAML Configuration

 Edit

Identifier (Entity ID)
Reply URL (Assertion Consumer Service URL)
Sign on URL
Relay State (Optional)
Logout Url (Optional)

6. The **Basic SAML Configuration** pane opens. Skip this section, because the app is preintegrated with Azure. Select **Save**.

The AWS application expects the SAML assertions in a specific format. You can manage the values of these attributes from the **User Attributes & Claims** section on the **Application integration** page.

7. On the **Set up Single Sign-On with SAML** page, select the **Edit** button.

2

Attributes & Claims

 Edit

givenname	user.givenname
surname	user.surname
emailaddress	user.mail
name	user.userprincipalname
Unique User Identifier	user.userprincipalname

8. In the **User Claims** section of the **User Attributes** pane, configure the SAML token attribute by using the values in the following table:

Name	Source attribute	Namespace
RoleSessionName	user.userprincipalname	https://aws.amazon.com/SAML/Attributes
Role	user.assignedroles	https://aws.amazon.com/SAML/Attributes
SessionDuration	"provide a value from 900 seconds (15 minutes) to 43200 seconds (12 hours)"	https://aws.amazon.com/SAML/Attributes

- a. Select **Add new claim** and then, on the **Manage user claims** pane, do the following:

User claims	
+ Add new claim Save Discard	
Name identifier value: user.userprincipalname	
CLAIM NAME	VALUE
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress	user.mail
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname	user.givenname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name	user.userprincipalname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier	user.userprincipalname
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname	user.surname

Manage user claims

* Name	<input type="text"/>
Namespace	<input type="text" value="Enter a namespace URI"/>
Source	<input checked="" type="radio"/> Attribute <input type="radio"/> Transformation
* Source attribute	<input type="text" value="Select from drop down"/>
<input type="button" value="Ok"/>	

- b. In the **Name** box, enter the attribute name.
- c. In the **Namespace** box, enter the namespace value.
- d. For the **Source**, select **Attribute**.
- e. In the **Source attribute** drop-down list, select the attribute.
- f. Select **Ok**, and then select **Save**.

(!) Note

For more information about roles in Microsoft Entra ID, see [Add app roles to your application and receive them in the token](#).

- 9. On the **Set up Single Sign-On with SAML** page, in the **SAML Signing Certificate** section, select **Download** to download the federation metadata XML file, and then save it to your computer.

3 SAML Signing Certificate

Status Active
Thumbprint <Thumbprintvalue>
Expiration <Expiration>
Notification Email <Email address>
App Federation Metadata Url <App Federation Metadata Url> 
Download Download
Certificate (Base64) 
Certificate (Raw) 
Federation Metadata XML 

Configure AWS SSO

1. In a new browser window, sign in to your AWS company site as administrator.
2. Select the **AWS Home** icon.



3. On the **AWS services** pane, under **Security, Identity & Compliance**, select **IAM (Identity & Access Management)**.

AWS services

Find a service by name or feature (for example, EC2, S3 or VM, storage).

Recently visited services



IAM

All services

Compute

EC2
EC2 Container Service
Lightsail
Elastic Beanstalk
Lambda
Batch

Developer Tools

CodeStar
CodeCommit
CodeBuild
CodeDeploy
CodePipeline
X-Ray

Storage

S3
EFS
Glacier
Storage Gateway

Management Tools

CloudWatch
CloudFormation
CloudTrail
Config
OpsWorks
Service Catalog
Trusted Advisor
Managed Services

Database

RDS
DynamoDB
ElastiCache
Redshift

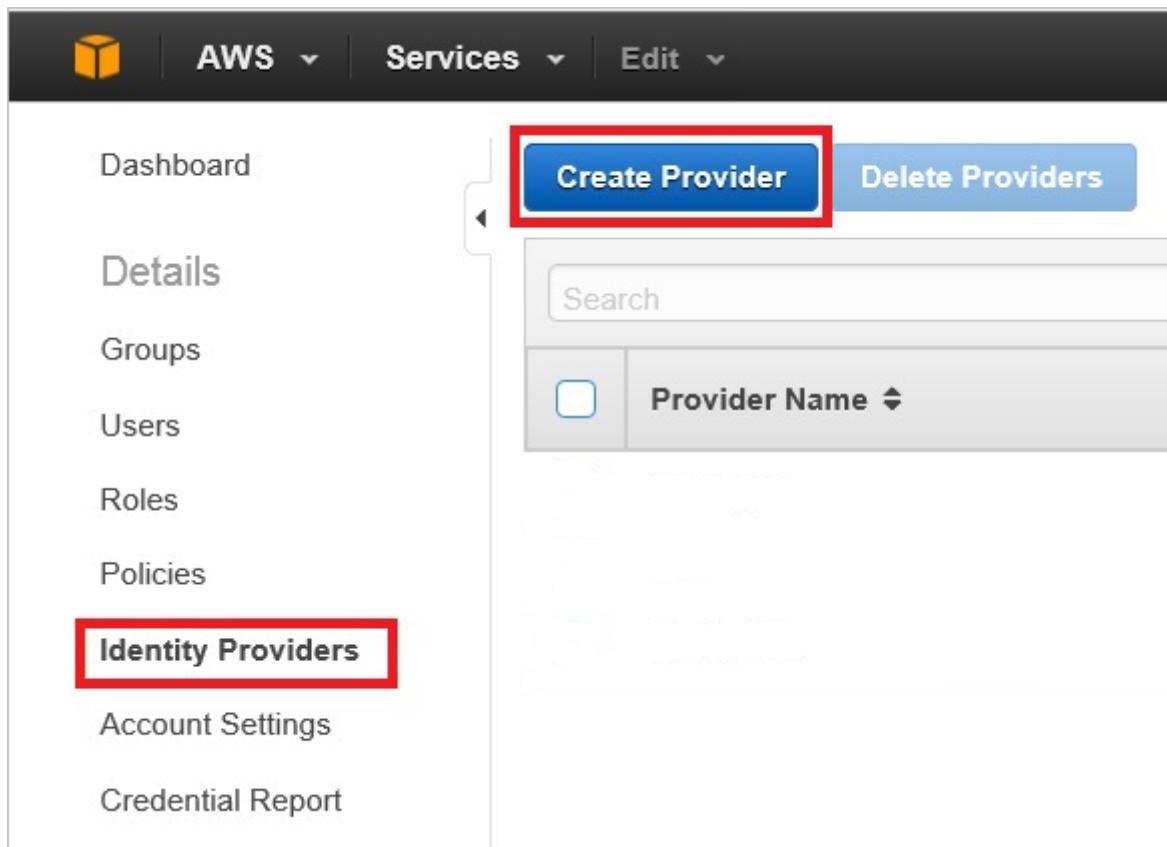
Security, Identity & Compliance

IAM
Inspector
Certificate Manager
Directory Service
WAF & Shield
Compliance Reports

Networking & Content Delivery

VPC
CloudFront
Direct Connect
Route 53

4. On the left pane, select **Identity Providers**, and then select **Create Provider**.



5. On the **Configure Provider** pane, do the following:

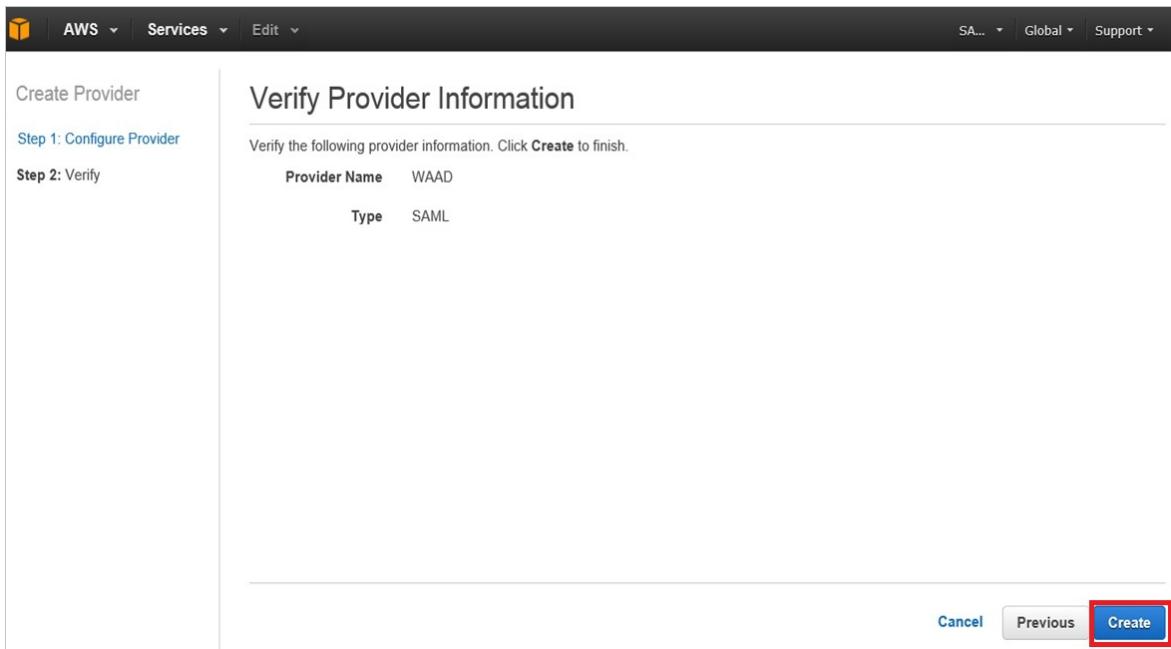
The screenshot shows the 'Configure Provider' step in the 'Create Provider' wizard. The left sidebar shows 'Step 1: Configure Provider' and 'Step 2: Verify'. The main pane is titled 'Configure Provider' and contains the following fields:

- Provider Type*: A dropdown menu set to 'SAML' (highlighted with a red box).
- Provider Name*: A text input box containing 'WAAD' (highlighted with a red box). A note below it says: 'Maximum 128 characters. Use alphanumeric and '-' characters.'
- Metadata Document*: A file input field with the path 'C:\fakepath\FederationMetadata' and a 'Choose File' button (highlighted with a red box).

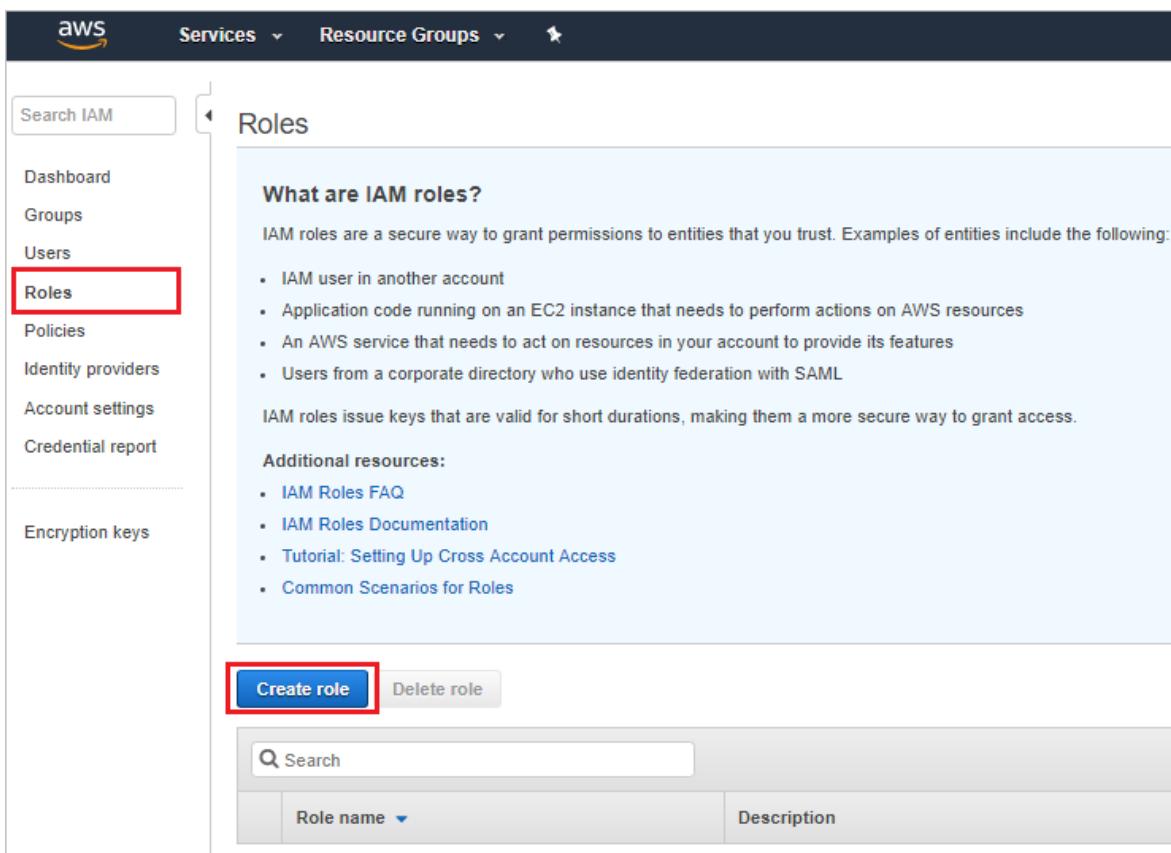
At the bottom, there is a note '* Required' and two buttons: 'Cancel' and 'Next Step' (highlighted with a red box).

- In the **Provider Type** drop-down list, select **SAML**.
- In the **Provider Name** box, enter a provider name (for example, **WAAD**).
- Next to the **Metadata Document** box, select **Choose File** to upload your downloaded federation metadata XML file to the Azure portal.
- Select **Next Step**.

6. On the Verify Provider Information pane, select **Create**.



7. On the left pane, select **Roles**, and then select **Create role**.



! Note

The combined length of the role Amazon Resource Name (ARN) and the SAML provider ARN for a role that's being imported must be 240 or fewer characters.

8. On the **Create role** page, do the following:

The screenshot shows the 'Create role' wizard at step 1. The title is 'Select type of trusted entity'. There are four options: 'AWS service' (EC2, Lambda and others), 'Another AWS account' (Belonging to you or 3rd party), 'Web identity' (Cognito or any OpenID provider), and 'SAML 2.0 federation' (Your corporate directory). The 'SAML 2.0 federation' option is highlighted with a red box. Below it, a note says 'Allows users that are federated with SAML 2.0 to assume this role to perform actions in your account.' A 'Learn more' link is provided. The next section is 'Choose a SAML 2.0 provider'. It shows a dropdown 'SAML provider' set to 'WAAD' (also highlighted with a red box), a 'Create new provider' button, and a 'Refresh' button. Two radio buttons are shown: 'Allow programmatic access only' (unchecked) and 'Allow programmatic and AWS Management Console access' (checked). Below this are fields for 'Attribute' (set to 'SAML:aud') and 'Value*' (set to 'https://signin.aws.amazon.com/saml'). A 'Condition' section with a 'Add condition (optional)' link is also present. At the bottom, there's a note '* Required', a 'Cancel' button, and a 'Next: Permissions' button (highlighted with a red box).

- a. Under **Select type of trusted entity**, select **SAML 2.0 federation**.
 - b. Under **Choose a SAML 2.0 provider**, select the SAML provider that you created previously (for example, *WAAD*)
 - c. Select **Allow programmatic and AWS Management Console access**.
 - d. Select **Next: Permissions**.
9. In the search box, enter **Administrator Access**, select the **AdministratorAccess** check box, and then select **Next: Tags**.

The screenshot shows the 'Filter policies' search bar at the top left with the text 'Administrator Access'. To the right, it says 'Showing 351 results'. Below the search bar is a table with two columns: 'Policy name' and 'Used as'. The 'AdministratorAccess' policy is selected, indicated by a checked checkbox in the first column and highlighted with a red border around the entire row. Other policies listed include 'AccessAnalyzerServiceRolePolicy', 'AlexaForBusinessFullAccess', 'AlexaForBusinessPolyDelegatedAccessPolicy', 'AlexaForBusinessReadOnlyAccess', 'AmazonAPIGatewayAdministrator', 'AmazonAPIGatewayInvokeFullAccess', and 'AmazonAppStreamFullAccess'. At the bottom of the table, there is a link 'Set permissions boundary'. Below the table, there are buttons for 'Cancel', 'Previous', and 'Next: Tags', with 'Next: Tags' also highlighted with a red border.

10. On the **Add tags (optional)** pane, do the following:

The screenshot shows the 'Add tags (optional)' pane. It contains a table with two columns: 'Key' and 'Value (optional)'. A single tag is listed: 'Name' in the Key column and '<Account Name> -aws-admin' in the Value column. There is a 'Remove' button next to the value. Below the table, there is a note: 'You can add 49 more tags.' At the bottom, there are buttons for 'Cancel', 'Previous', and 'Next: Review', with 'Next: Review' highlighted with a red border.

- a. In the **Key** box, enter the key name (for example, *Azureadtest*).
- b. In the **Value (optional)** box, enter the key value in the following format: `<accountname-aws-admin>`. The account name should be in all lowercase letters.
- c. Select **Next: Review**.

11. On the **Review** pane, do the following:

Review

Provide the required information below and review this role before you create it.

Role name*	<Account Name> -aws-admin
Use alphanumeric and '+=_@-' characters. Maximum 64 characters.	
Role description	<Account Name> -aws-admin
Maximum 1000 characters. Use alphanumeric and '+=_@-' characters.	
Trusted entities The identity provider(s) arn:aws:iam::757365230471:saml-provider/WAAD1	
Policies	AdministratorAccess
Permissions boundary Permissions boundary is not set	
* Required Cancel Previous Create role	

- In the **Role name** box, enter the value in the following format: <accountname-aws-admin>.
- In the **Role description** box, enter the value that you used for the role name.
- Select **Create role**.
- Create as many roles as you need, and map them to the identity provider.

(!) Note

Similarly, you can create other roles, such as *accountname-finance-admin*, *accountname-read-only-user*, *accountname-devops-user*, or *accountname-tpm-user*, each with a different policy attached to it. You can change these role policies later, according to the requirements for each AWS account. It's a good idea to keep the same policies for each role across the AWS accounts.

- Be sure to note the account ID for the AWS account either from the Amazon Elastic Compute Cloud (Amazon EC2) properties pane or the IAM dashboard, as shown in the following screenshot:

Role name	Description
ADFS-Admin	Administrator

- Sign in to the [Azure portal](#), and then browse to **Groups**.

14. Create new groups with the same name as that of the IAM roles you created earlier, and then note the value in the **Object Id** box of each of these new groups.

The screenshot shows the Microsoft Entra ID interface for managing groups. The left sidebar has sections for Overview, Manage (Properties, Members, Owners, Group memberships, Applications, Licenses, Azure resources), Activity (Access reviews, Audit logs), and Troubleshooting + Support (Troubleshoot, New support request). The main area displays the 'master-aws-admin' group details. It includes a large purple 'MA' logo, the group name 'master-aws-admin', and the object ID '25e490c7-7cd0-487e-b2ee-587e366393b1'. Below this are fields for Membership type (Assigned), Source (Cloud), Type (Security), and a 'Members' section showing 0 User(s), 0 Group(s), 0 Device(s), and 0 Other(s). At the bottom are sections for Group memberships (0) and Owners (0).

15. Sign out of the current AWS account, and then sign in to another account where you want to configure SSO with Microsoft Entra ID.

16. After you've created all the roles in the accounts, they're displayed in the **Roles** list for those accounts.

Search		
Role name	Description	Trusted entities
<input type="checkbox"/> [REDACTED]	Administrators	Identity Provider: arn:aws:iam::[REDACTED]
<input type="checkbox"/> [REDACTED]	Group of auditors	Identity Provider: arn:aws:iam::[REDACTED]

You next need to capture all the role ARNs and trusted entities for all roles across all accounts. You'll need to map them manually with the Microsoft Entra application. To do so:

1. Select each role to copy its role ARN and trusted entity values. You'll need them for all the roles that you'll create in Microsoft Entra ID.

Summary

Role ARN	arn:aws:iam::[REDACTED]:role/Auditors [i]
Role description	Group of auditors Edit
Instance Profile ARNs	[REDACTED]
Path	/
Creation time	2018-04-11 13:44 PDT
Maximum CLI/API session duration	1 hour Edit

Permissions Trust relationships Access Advisor Revoke sessions

You can view the trusted entities that can assume the role and the access conditions for the role. [Show policy document](#)

Edit trust relationship

Trusted entities

The following trusted entities can assume this role.

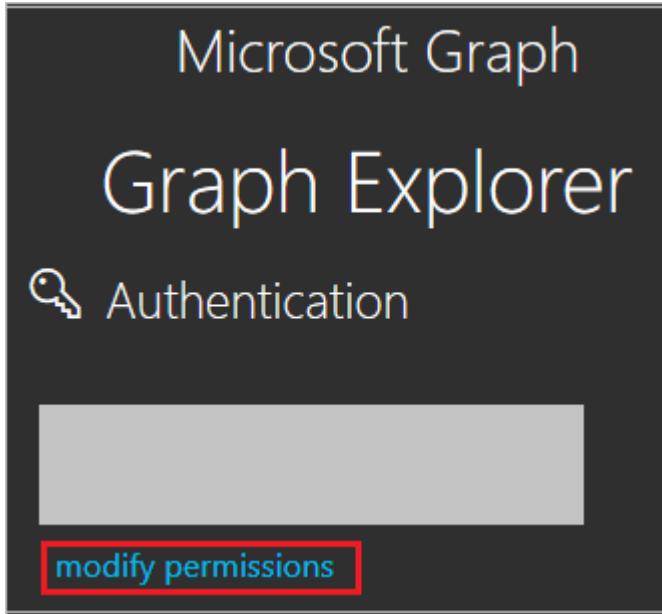
Trusted entities
arn:aws:iam::[REDACTED]:saml-provider/WAAD1

Conditions

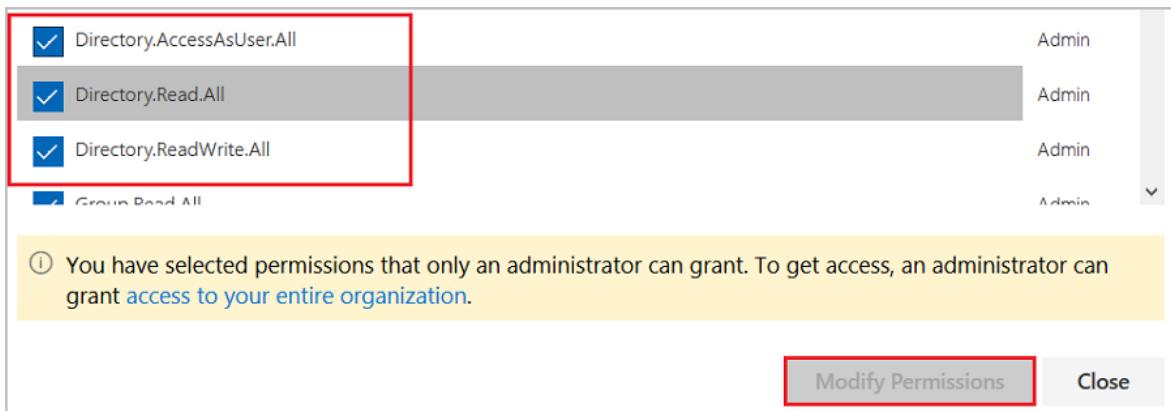
The following conditions define how and when trusted entities can assume the role.

Condition	Key	Value
StringEquals	SAML:aud	https://signin.aws.amazon.com/saml

2. Repeat the preceding step for all the roles in all the accounts, and then store them in a text file in the following format: <Role ARN>,<Trusted entities>.
3. Open [Microsoft Graph Explorer](#), and then do the following:
 - a. Sign in to the Microsoft Graph Explorer site with the Global Administrator or Co-admin credentials for your tenant.
 - b. You need sufficient permissions to create the roles. Select **modify permissions**.



- c. In the permissions list, if you don't already have the permissions that are shown in the following screenshot, select each one, and then select **Modify Permissions**.



- d. Sign in to Graph Explorer again, and accept the site usage conditions.
- e. At the top of the pane, select **GET** for the method, select **beta** for the version, and then, in the query box, enter either of the following:
- To fetch all the service principals from your tenant, use <https://graph.microsoft.com/beta/servicePrincipals>.
 - If you're using multiple directories, use <https://graph.microsoft.com/beta/contoso.com/servicePrincipals>, which contains your primary domain.

```

{
  "@odata.context": "https://graph.microsoft.com/beta/$metadata#servicePrincipals/$entity",
  "id": "8164e784-8a01-46c9-89fd-3076bd9656d0",
  "deletedDateTime": null,
  "accountEnabled": true,
  "appDisplayName": "",
  "appId": "8e1d26f3-9519-4d66-8577-65fa3261c7ff",
  "appOwnerOrganizationId": "0ac53016-3006-4227-9eeb-89d63f8055b6",
  "appRoleAssignmentRequired": true,
  "displayName": ""
}
  
```

Success - Status Code 200, 6146ms

```

{
  "@odata.context": "https://graph.microsoft.com/beta/$metadata#servicePrincipals",
  "@odata.nextLink": "https://graph.microsoft.com/beta/servicePrincipals?$skiptoken=X%27445370740200010000035536572766963655072696E636970616C5F31643466396561652D32",
  "value": [
    {
      "id": "00008fa9-7197-4f17-b74b-cf702e697ddc",
      "deletedDateTime": null,
      "accountEnabled": true,
      ...
    }
  ]
}
  
```

- f. From the list of service principals, get the one you need to modify.

You can also search the application for all the listed service principals by selecting Ctrl+F. To get a specific service principal, include in the query the service principal object ID, which you copied earlier from the Microsoft Entra Properties pane, as shown here:

<https://graph.microsoft.com/beta/servicePrincipals/<objectID>>.