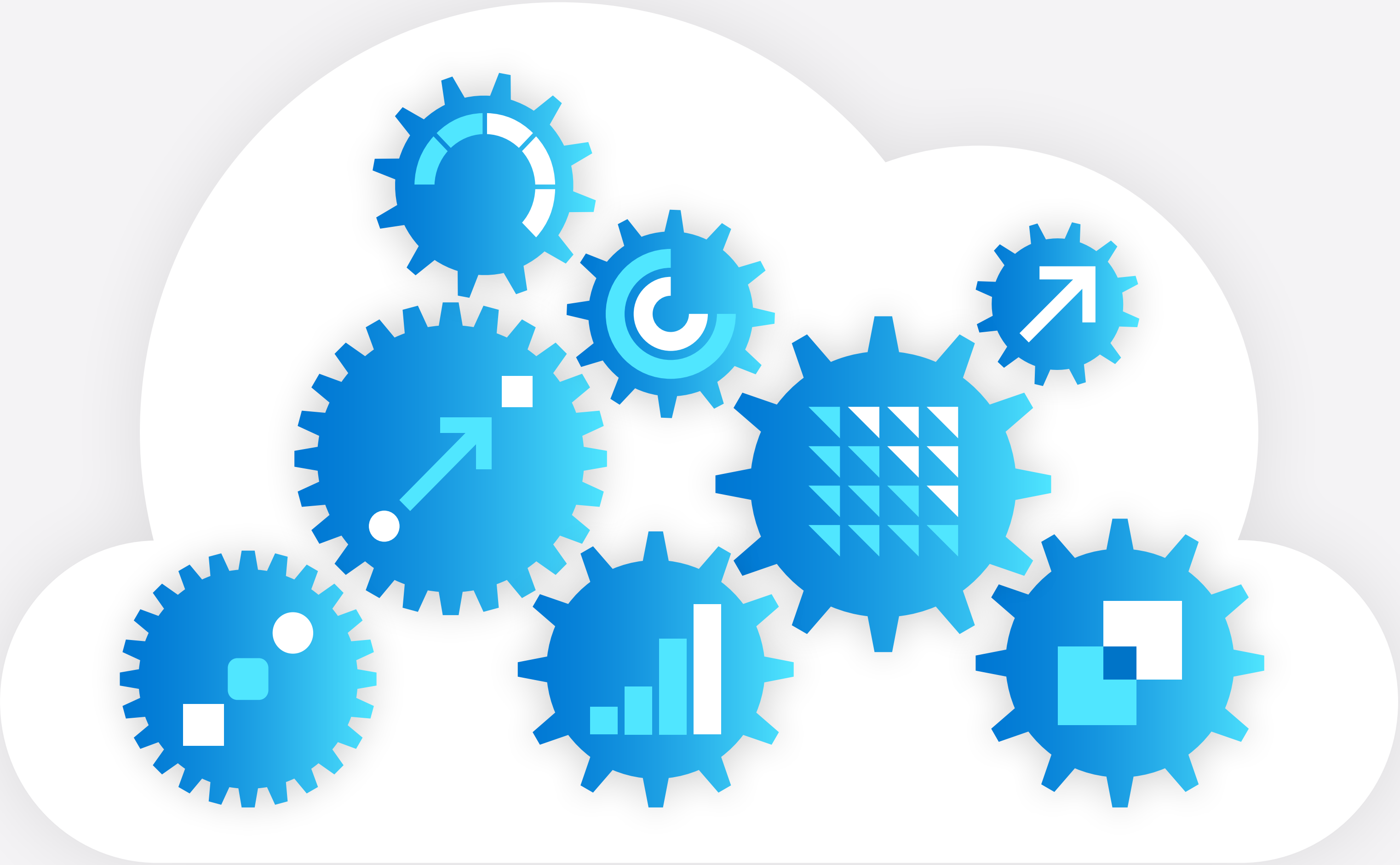
E-book series

Microsoft Azure

Modernize Your Applications with Azure Spring Apps



Modernize Your Applications with Azure Spring Apps

Section 1

Introduction to application modernization

Businesses need to modernize applications to stay up to date, save costs, and optimize resources, allowing them to focus on their core business goals and priorities.

The goal of modernization is to enhance the quality of software, boost performance, and improve the overall customer experience. Modernization also involves taking existing legacy applications and digitally transforming their platform infrastructure, architecture, or features to meet modern business demands.

Application modernization, specifically, is the process of updating an organization's application estate. This involves improving application lifecycle management and data management through a cloud-native approach.

As time moves on, every business will find they have a natural need to modernize. The only question is how. In this e-book, you'll learn how to approach modernization using Azure Spring Apps. Azure Spring Apps is a platform as a service (PaaS) solution by Azure that fully manages application development with built-in service discovery and configuration management. Enterprises can use Azure Spring Apps as a central part of their application modernization process. Azure Spring Apps makes implementation and management easy so businesses can focus on business goals.

This guide covers best practices and how to choose the right tools for modernization, as well as setting up your organization for success post-migration. Whether you're a technical decision maker or part of an application team, this e-book has the foundational knowledge you need to modernize with Microsoft Azure Spring Apps.

When to modernize

Most enterprises today have a vast suite of applications in their environment. These applications have been developed over a period or have been procured off the shelf from various software providers. Each of these applications caters to a specific business unit, such as finance, payroll, or even customer-facing functions, such as CRM tools. Over time, technology evolves, and so do company business models. Some applications that suited business needs for an earlier stage in the growth of a business may no longer be required. In contrast, another group of applications might be needed in a different form than they were initially conceived. As an example, we'll examine Norway's Komplett Group and their reason for pursuing a successful migration to the cloud.

IT teams look to modernize applications like Komplett's—ones that have become outdated due to moving business needs—to better fit current and future requirements. In some cases, this involves rearchitecting applications and reworking code directly. In other scenarios, teams might look to take advantage of PaaS and other cloud-native solutions, such as Azure Spring Apps.

PaaS solutions manage the underlying components required in an enterprise application, enabling teams to focus on innovation and delivering business logic. This shifts the work of necessary administration and maintenance tasks away from developers to the PaaS provider, freeing development teams to focus on faster delivery of business value.

Delivering e-commerce success at scale

Komplett Group is a household name for 1.8 million e-commerce customers throughout Scandinavia. With 10 web stores located across the Nordic region, Komplett's business model connects product manufacturers with a variety of customers—from consumers and businesses to resellers and public enterprises.

In the past, Komplett relied on its on-premises datacenters to operate and fulfill a complex e-commerce operation. This was not only expensive and time-consuming to maintain but also limited in its ability to scale at speed. So, Komplett decided to explore cloud offerings, attracted to the cloud's ability to improve service while reducing costs. "We were looking for a platform we could grow with," said Thomas Wilhelmsen, Head of IT Operations for Komplett. "For us, Azure was the perfect platform to transition from the datacenter to the public cloud."

The cloud migration journey

The company moved almost everything to Azure, eliminating the complexity of virtual machines and going straight to Azure App Services environments. But the ultimate test of Komplett's e-commerce operations would come during the busy holiday season, when the company handled up to 30 times its typical daily volume of web traffic on a single day—Black Friday.

Before migrating to the cloud, capacity planning for such peak events meant buying physical hardware, including additional servers and software licenses, to handle an exponential increase in website visitors. Boosting capacity also required the extra space and resources to house and power the new equipment.

Thanks to Azure's PaaS offerings, Komplett's ability to scale quickly was immediate and invaluable. Komplett found it could scale up or down on a daily basis if needed rather than being fully provisioned 24 hours a day, 7 days a week. The speed of development and deployment on the Azure platform also proved to be a game changer.

"We saw more than 500 views a second and delivered 21 TB of data to our customers, with great response times," Wilhelmsen said. "Using Azure, we were easily able to scale on demand to get the performance we needed."

Benefits of application modernization

Modernizing your applications comes with several benefits:

- **Modernization helps developers deliver new features to their customers faster:** A modernized application on the cloud is much easier to manage than its on-premises counterpart. Unlike an on-premises platform, a managed PaaS platform automatically performs many manual administration tasks to which the team would typically have to devote time and energy. This enables your team to focus on delivering new experiences.

- **Modernization improves scalability:** By shifting to a cloud-based platform, it's easier to handle changes in traffic without over-investing in on-premises hardware. The Azure cloud provides scalability to allow an application to manage and handle traffic increases seamlessly compared to its on-premises counterpart.

- **Modernization improves security:** Security benefits from using a host on the cloud since the PaaS provider adds their security practices and defenses to yours. The Azure cloud adds its own pillars of security to any application, ensuring it is protected from attacks.

- **Modernization brings down costs:** Applications that have been modernized to fit the requirements of the business more closely can almost immediately bring down the overall cost of ownership. Organizations can derive the most value from modernization simply by ensuring their applications align with their evolving business goals.

- **Modernization provides flexibility:** In a changing business environment, microservices architecture can provide the flexibility required to ensure that applications are always ready to adapt. Businesses can deliver new functionality to their users through containerization technologies such as Azure Spring Apps and Azure Kubernetes Service (AKS).

Any organization considering modernization is best served by exploring various technologies that can be used for application modernization to identify which option would be best for their specific use case.

Choosing the right app modernization services

Enterprise teams evaluate different technologies depending on their business requirements. Teams wanting to improve day-to-day manageability might choose PaaS solutions such as Azure App Service as the primary approach since, in this case, Azure takes care of the entire underlying infrastructure and middleware components.

Teams that might be looking to expose their application back end to external sources can look at Azure API Management as their approach of choice since it offers a single gateway to manage all their needs. Similarly, AKS and Azure Spring Apps can be great choices for microservices and container-based approaches. Azure Spring Apps is built on top of AKS and gives a fully managed infrastructure to teams who might want to deploy their Spring applications on it.

We'll use Azure services as our frame of reference, but whichever service you choose, it will fall into one of the following four types:

1. PaaS solutions, such as Azure App Service

2. API management, such as Azure API Management

3. Microservices and container support, such as AKS

4. Fully managed PaaS solutions, such as Azure Spring Apps

We'll go through each type, using Azure technologies as examples.

PaaS services

A PaaS solution, such as Azure App Service, can build the underlying components of an architecture, such as the operating system, runtime, and middleware. This allows teams to focus on code while the PaaS takes care of necessary maintenance tasks, such as operating system patching. This fosters better productivity, as your team now focuses on delivering newer features and bringing them to market faster. At the same time, the PaaS manages your key concerns around availability, scalability, and security.

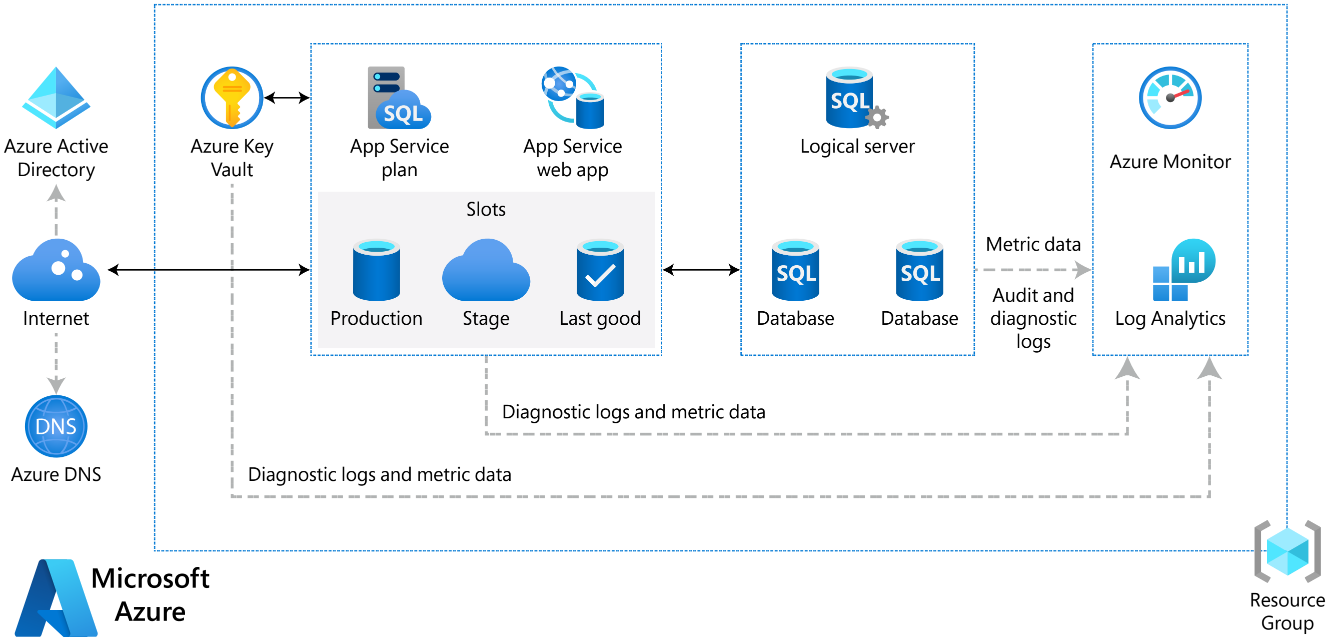


Figure 1: Application deployed over Azure App Service

Figure 1 represents a standard reference architecture for Azure App Service, including the application, its staging environments, monitoring and logging systems (in this case, Azure Monitor), and the database. Azure App Service is the popular choice for a modernization tool since this approach involves little to no code change in most cases.

If the runtime is supported in Azure, developers can simply deploy the code to Azure App Service. In organizations with no robust DevOps practices, developers can ship code and transfer it via FTP to their Azure App Service plan. The team can then utilize deployment slots to perform zero-downtime deployments, with no complex operations required to be carried out by your release teams.

Any PaaS needs to be able to monitor performance and keep meaningful logs. Azure App Service can use its native tools, such as Azure Monitor and Application Insights, to monitor the performance of the application logs and metric data without the overhead of managing it. Azure App Service is one of the fastest ways to modernize a legacy application. This approach almost instantly eliminates all the maintenance activity that you require to keep the application up and running.

> **We were looking for a platform we could grow with. For us, Azure was the perfect platform to transition from the datacenter to the public cloud.”**
> **Thomas Wilhelmsen, Head of IT Operations for Komplett**

API management services

API management services, such as Azure API Management, are platforms that enable teams to manage their APIs through a single gateway. Teams looking for a single, unified management experience across all their APIs will find that an API management service makes it easy. Azure API Management supports APIs not only on Azure but also on third-party clouds and on-premises datacenters.

Azure API Management offers full observability across all internal and external APIs. Teams can build products and deliver value to customers through an API-first approach, and APIs can be monetized when required. Good API management fosters the decoupling of front-end and back-end resources, and the APIs can be managed separately outside the lifecycle of back-end systems.

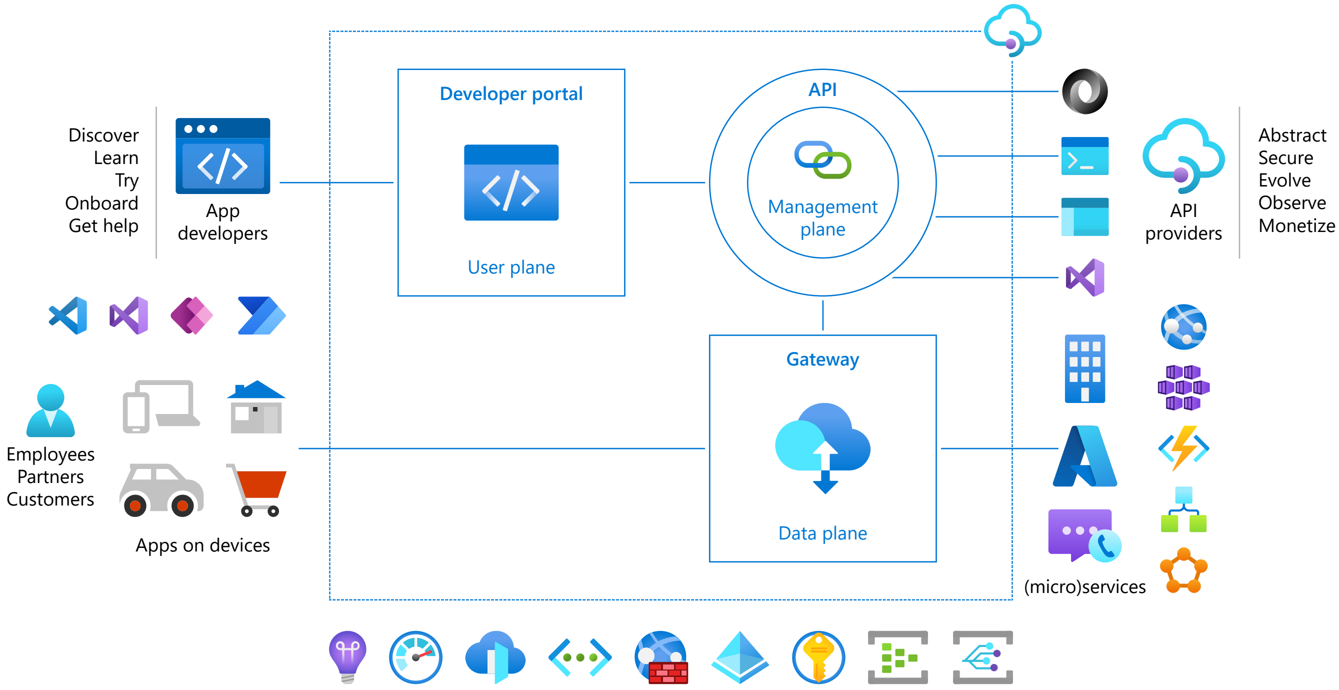


Figure 2: An Azure API Management diagram

The best API management tools offer clear documentation, security, rate-limiting tools, and the ability to handle multiple requests. Azure API Management provides the following benefits:

- External developers can learn how to build integrations and mock responses through the documentation portal.
- A robust security mechanism means that teams can block traffic from an IP address—immediately identifying suspicious requests and blocking them before they can cause any damage.
- Rate limiting ensures that requests from a particular source do not drown your API Management platform.
- All back-end resource requests can be handled through **Azure Active Directory (Azure AD)** or other means, such as JSON Web Tokens.
- Multiple versions of APIs can be launched and hosted in parallel without maintaining a separate codebase in the back end.

Teams that are looking to expose their applications through API endpoints that can be consumed by internal or external clients will find that an API management tool is essential. Azure API Management provides robust scalability and enterprise-grade security, which allows developers to focus on building business logic. At the same time, the entire administration and maintenance workflow can be owned and managed by Azure.

Container platforms

Many applications are now packaged in the form of Docker containers. Building and running an application in a single container can be easy; however, the complexity grows when the application itself consists of multiple Docker containers running over multiple servers. This is where a container orchestration platform, such as AKS, a managed Kubernetes platform from Microsoft, comes in. Many legacy applications are refactored into microservices, which can then be hosted on a Kubernetes cluster.

AKS can manage both stateful and stateless applications. You can build portable microservices-based applications and use AKS to manage their availability and orchestration. Applications written in any programming language, for example, Java, .NET, or Python, can be deployed on AKS.

Applications can be deployed using a declarative syntax model. Deployment configurations are written in YAML format and submitted to the Kubernetes cluster. The cluster can then make decisions on how to host the application. Any existing DevOps tool, such as Azure DevOps or GitHub, can be used to coordinate and manage your releases on AKS.

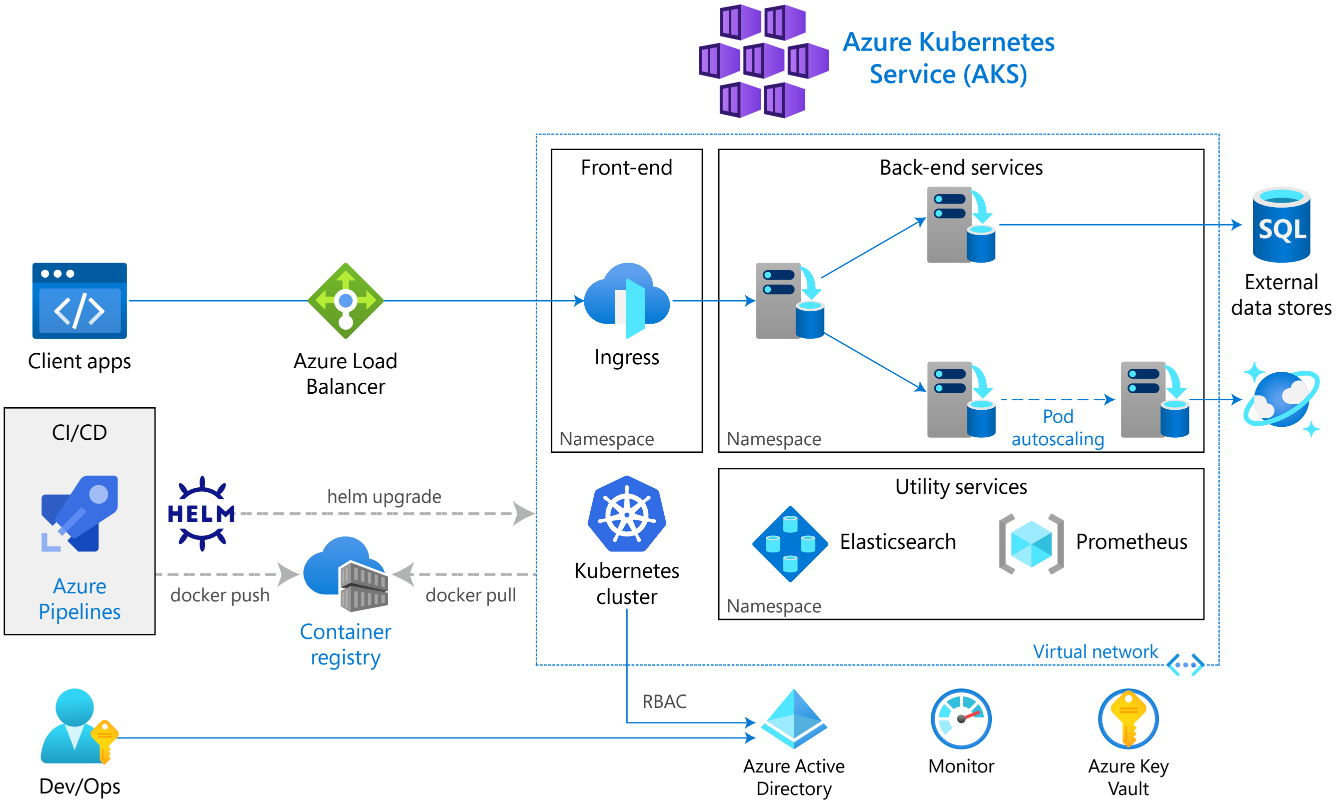


Figure 3: Standard Kubernetes cluster with deployments and DevOps pipeline

Figure 3 represents a basic AKS cluster. An AKS cluster consists of nodes where the actual deployments or the application workloads are hosted and run. In the event of a surge in traffic, the cluster can spin up additional worker nodes or VMs through the cluster autoscaler.

In the case of a horizontal pod autoscaler (HPA), a number of replica pods are distributed across the existing set of worker nodes. AKS performs well on scalability; however, it still leaves a lot of infrastructure to be managed by the customer themself.

AKS may be right for you if you have the skill set to configure and manage Kubernetes containers in-house. For Spring applications, this is where offerings such as Azure Spring Apps help with fully managed platform services.

Fully managed PaaS services

Fully managed PaaS services bring together a PaaS solution along with development management and containerization built in. For example, Azure Spring Apps is a fully managed PaaS service with built-in application lifecycle management. Developers can use Azure Spring Apps to host their Spring Boot and Steeltoe .NET core apps and make them more cloud-native. This helps them to focus on the code without bothering themselves with the underlying infrastructure.

Under the hood, Azure Spring Apps runs on the Kubernetes infrastructure; however, teams can host their applications over it without having to learn how to manage Kubernetes. This can help a team to get up and running with their microservices applications within a very short period of time.

Teams can struggle to manage the application lifecycle along with managing the underlying infrastructure for Spring Boot applications. On Azure Spring Apps, developers can also use Spring Cloud components to coordinate between microservices.

Components such as the config server, service registry, and log stream are built-in and native to the platform. Once modernized, the applications can easily be monitored from Azure itself without having to integrate any other additional service. Spring Boot applications require no code changes for running on Azure Spring Apps.

> **We saw more than 500 views a second and delivered 21 TB of data to our customers, with great response times. Using Azure, we were easily able to scale on demand to get the performance we needed.”**
> **Thomas Wilhelmsen, Head of IT Operations for Komplett**

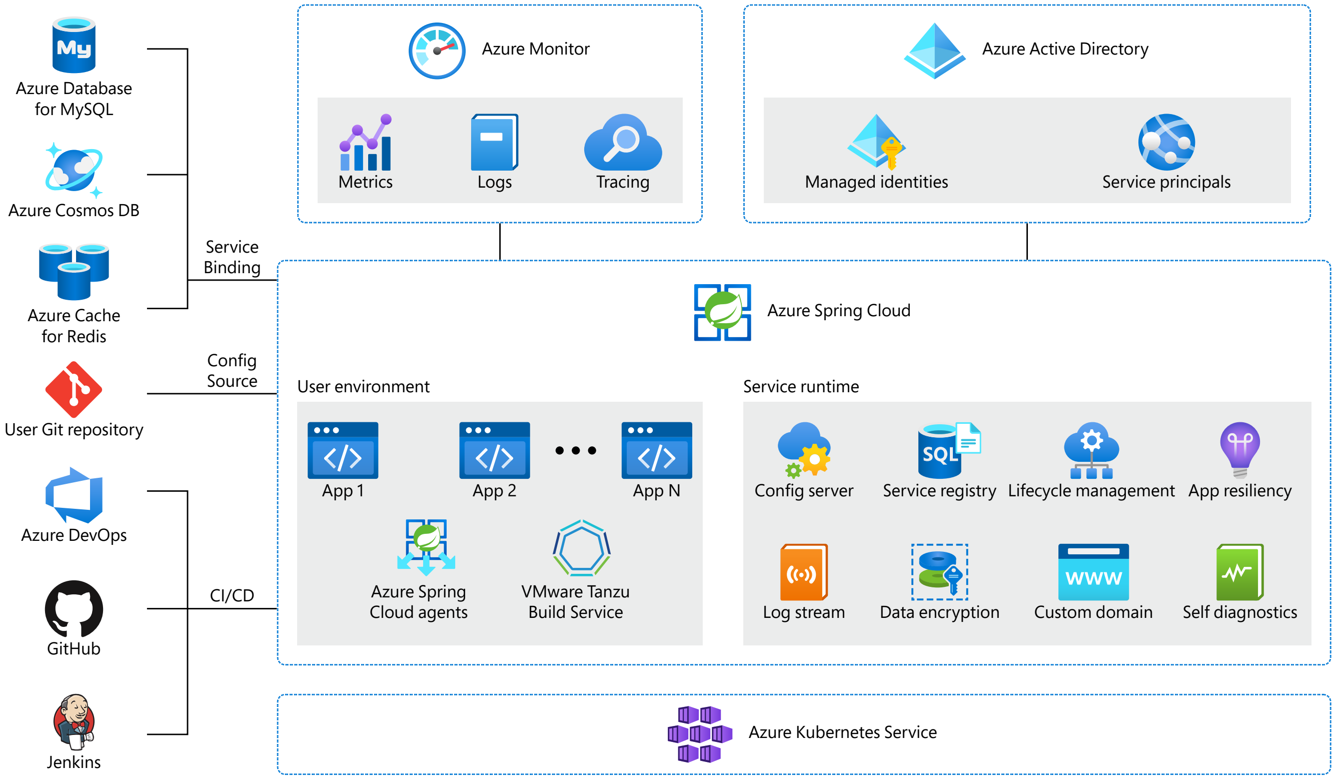


Figure 4: Azure Spring Apps with a deployment, monitoring, and identity ecosystem

Azure Spring Apps comes with workloads and service runtime component features such as distributed tracing and a circuit breaker available out of the box, and they need not be implemented separately. Scalability for Azure Spring Apps can be achieved through simple options available on the platform itself. Every single request can be traced, which makes troubleshooting the application much easier. The entire application lifecycle can be managed through the service, reducing the need to manage the application manually and helping teams bring applications to the market faster.

Modernization can often be a vast discussion and can mean different things to different teams. However, the choice of modernizing an application is often driven by changes in business requirements. Applications are required to be updated to suit their current and future use cases.

Modernization often involves breaking the application into microservices so it can run efficiently on containers. This is where services such as AKS, Azure Spring Apps, and also Azure API Management can bring value to businesses. Teams for which manageability is a key concern should consider fully managed platform services such as Azure Spring Apps.

Section 2

Steps to modernization

Azure Spring Apps takes on the responsibility of the underlying layers, enabling the development team to focus on IP creation. All components of application lifecycle management can now be managed by Azure Spring Apps itself. This reduces the overall development time, thereby improving developer velocity. Azure Spring Apps brings the focus back to feature development while taking care of the Spring components, such as Config Server. With reduced workloads for the team, organizations can improve their capability to deliver applications faster.

Organizations looking to modernize their applications should start by identifying the features required in the new platform. This involves deciding what the application is expected to deliver or achieve. Once the requirements have been defined, the team can evaluate the best approach for their use case.

Migration is always easiest when an app is well-architected and created according to best practices. To keep migration smooth, it's important to know the available guidelines to help developers design an efficient and reliable architecture.

Azure Spring Apps best practices and reference architecture

An Azure Spring Apps architecture should be developed with careful design consideration of the Microsoft Azure Well-Architected Framework. The Well-Architected Framework is a set of best practices and guidelines that help teams design an architecture while keeping in mind the five pillars of architectural excellence:

These pillars are a guide to evaluating a design's potential for success. A well-designed architecture should try to excel on all of these fronts.

The Well-Architected Framework is not a rule but rather a set of principles on which the success of your architecture can be evaluated. Most teams look at these same pillars while designing an application architecture on Azure to meet their needs.

Let's discuss the pillars of the Microsoft Azure Well-Architected Framework in a bit more detail and see how each of these pillars contributes toward a resilient application architecture on Azure.

Reliability

Azure Spring Apps is built on the underlying AKS engine. This means the principles of scalability and reliability for AKS clusters are also applicable to Azure Spring Apps. Microsoft recommends using a hub and spoke design to help increase the availability of the application in terms of the failure of individual services.

The architecture can be scaled up to deploy it to multiple regions. For the public application use case, Azure Front Door and Azure Application Gateway ensure availability. In contrast, a private Domain Name System (DNS) can provide access whenever failover is required for internal applications.

Security

The application architecture should comply with standard industry-defined controls and benchmarks. Design principles of identity, access management, and storage must be considered while building the architecture. Architecture should always follow the principle of least privilege.

Microsoft recommends a few best practices and benchmarks, such as the Cloud Control Matrix. This includes a series of mappings and implementation guidelines for application security. In addition, Microsoft provides-guidance and best practices for implementing solutions on Azure. The team should look at implementing the Azure Security Benchmark for application and infrastructure security.

Cost optimization

Cost is always an important point to consider when designing an application. Teams should look to efficiently utilize their resources to get maximum value from their investments. This includes computing resources and operational overheads that application deployments bring along.

Teams can plan and deploy multiple applications of varying sizes to share resources of a single instance of Azure Spring Apps. The instance supports the autoscaling of applications to improve utilization and cost efficiency. Services such as Azure Monitor and Application Insights can help lower operational costs. Using insights from these solutions, you can implement automation to scale the components of the system in real-time.

Operational excellence

Azure Spring Apps can be complemented with Azure cloud components to build a reliable architecture that is much more manageable than its on-premises counterpart. DevOps pipelines can ensure that manual errors are eliminated from deployments. The inbuilt monitoring capabilities of Azure can make it very easy to identify any errors with the application. Finally, the security posture can be improved by using Azure WAF and Microsoft Defender for Cloud to analyze the data.

Performance efficiency

Performance efficiency is the ability of your workload to scale to meet the demands users place on it in an efficient manner. In the past, good performance efficiency would require the provision of enough capacity for peak usage; in the modern era, the main way to achieve performance efficiency is to use scaling appropriately and implement PaaS offerings that have scaling built in.

These pillars can be easier to understand by looking at an example, so in the next section, we'll look at a well-architected Azure Spring Apps reference architecture.

Azure Spring Apps reference architecture

The following Azure Spring Apps reference architecture is designed keeping in mind the architectural tenets and best practices of the Microsoft Azure Well-Architected Framework.

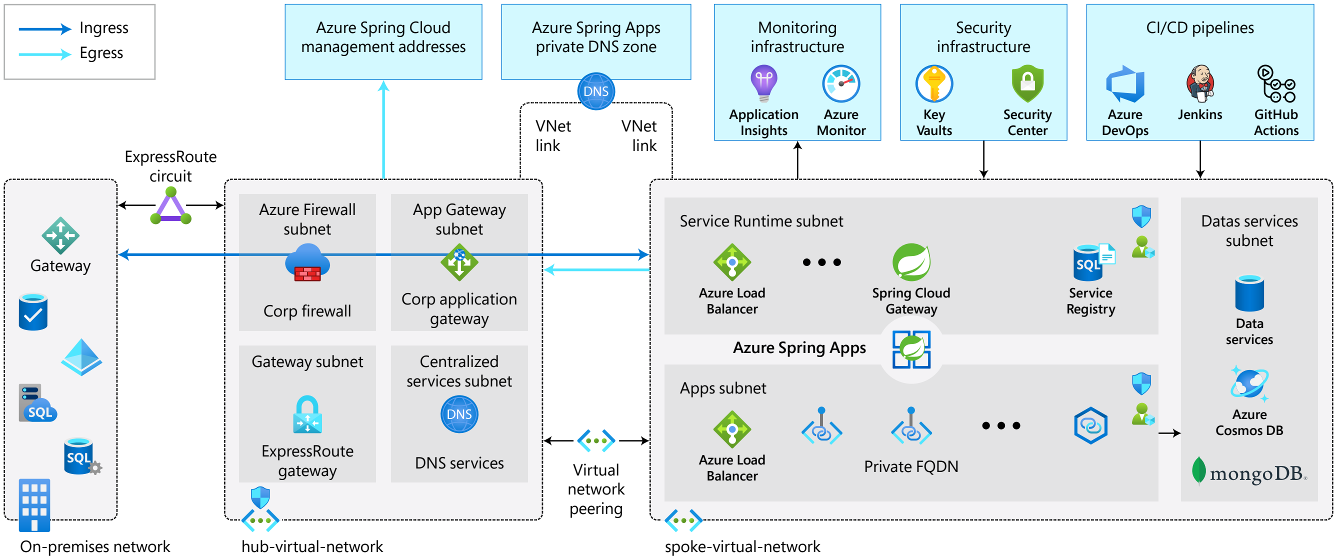


Figure 5: Azure Spring Apps reference architecture in the Standard tier

Figure 5 shows a foundation using a typical enterprise hub and spoke design for the use of Azure Spring Apps. In this design, Azure Spring Apps is deployed in a single spoke, dependent on shared services hosted in the hub.

The Standard tier comprises the Spring Cloud Config Server, Spring Cloud Service Registry, and kpack build services. These are standard Spring components that enable microservices to interact and communicate with each other, which is essential for microservices to perform their duties.

Let us also review some key Azure services used other than Azure Spring Apps components:

Azure WAF

Serves as the corporate firewall for all ingress traffic to the application.

Application Insights and Azure Monitor

Responsible for monitoring application availability and performance.

Key Vault and Security Center

Both services are part of the security infrastructure. Key Vault holds application secrets, while Azure Security Center offers unified security management and threat protection.

Continuous integration/ continuous deployment (CI/CD) services

Part of the deployment infrastructure, deployments can be done through any DevOps tool, such as Jenkins, Azure DevOps, or GitHub.

The Spring Apps standard architecture serves as a benchmark for applications running on the Standard tier utilizing standard Spring components.

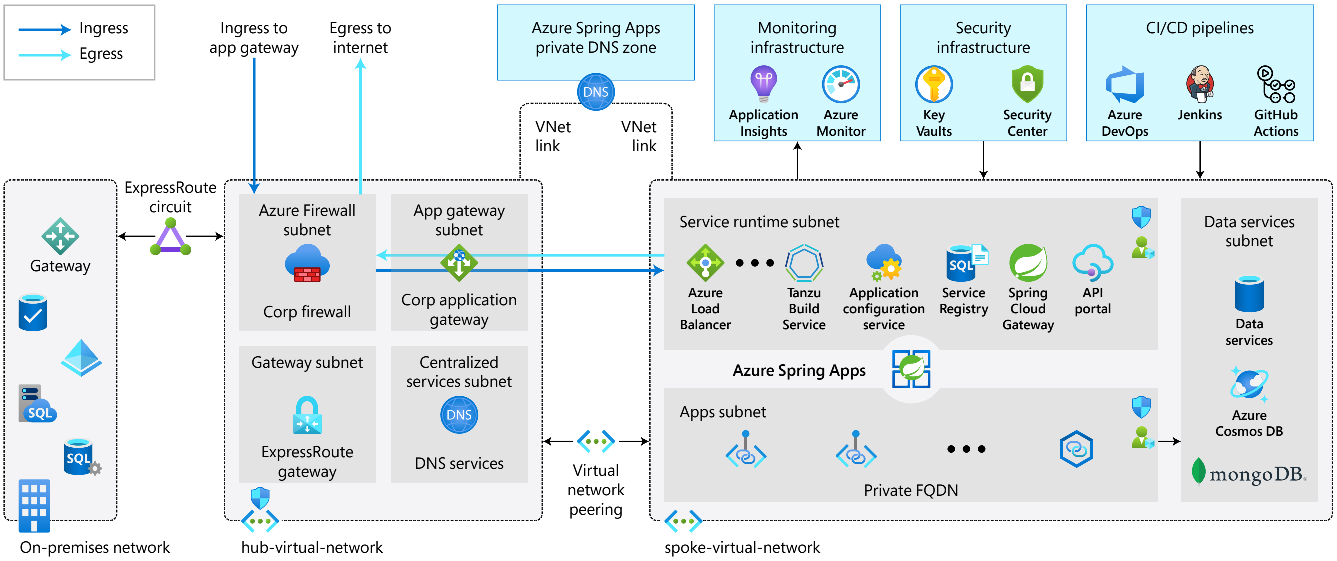


Figure 6: Azure Spring Apps reference architecture for the Enterprise tier

As you can see in *Figure 6*, the Enterprise tier of Azure Spring Apps is composed of VMware Tanzu service components, such as VMware Tanzu Build Service, Application Configuration Service for VMware Tanzu, VMware Tanzu Service Registry, Spring Cloud Gateway for VMware Tanzu, and the API portal for VMware Tanzu.

These are components designed and built by VMware for applications that require high reliability in their architecture. This is a great option for large enterprise applications running on Azure Spring Apps.

Teams migrating applications over to a service such as Azure Spring Apps should carry out the following steps before the applications are ready to be hosted.

Pre-migration

Pre-migration involves a series of actions and ideas that need to be considered by engineering teams as they embark on the journey of using Azure Spring Apps. This involves design-based guidelines, which simply help with the migration and hosting of their application on Azure. This is true for both brownfield and greenfield projects.

Application code changes

Certain changes to the existing application code may be required during pre-migration to make the application stateless. These changes aim to remove all state information from the code and store it separately, including any operating system-level dependencies.

Separating the data from the logic in the code is the best practice for creating reusable units of code and is necessary to get the most out of a PaaS system such as Azure Spring Apps. Azure Spring Apps offers specific versions of Java, Spring Boot, and Spring Cloud.

Some example code changes might be:

- If the application is storing state, then the code must be changed to store state outside the application, typically in a database or a caching system, such as Azure Cosmos DB, Azure SQL, or Azure Cache for Redis.

- If the application contains any operating system-specific code, those dependencies must be removed. Any static content posted by the application on the local file system can be moved to services such as Azure Blob storage.

- Application aspects such as external data providers, brokers, and identity providers such as OAuth2 need to be reviewed to identify any dependencies for migration. For Spring Cloud, determine whether your application has explicit dependencies on Zipkin. External clients communicating through ports other than 80 and 443 will have to be modified.

Application configuration changes

Certain applications might require changes to their existing custom configuration. This involves standardizing certain aspects, such as the application ports, dependencies, and secret management.

Some key configuration points to consider are:

- Azure Spring Apps can be integrated with Azure Monitor for performance management and to monitor any changes in the application.
- All communication must take place through these standard ports, such as 80/443.
- Azure Spring Apps should be given access to Spring Cloud Config Server data, which is stored on-premises.
- In addition to the previous points, compute requirements for the application need to be considered to run the application on Azure Spring Apps.
- The application can use Azure Key Vault for secrets. You can also configure Spring Cloud Vault to access secrets.
- Check application identity providers to identify any dependencies required during migration.

Pre-migration activities assess and review existing application capabilities. They identify key areas where changes are required before the application can be migrated to Azure Spring Apps. Once the set of pre-migration actions has been completed, what is left is a standardized application that can now easily be migrated to Azure Spring Apps.

Pre-migration activities introduce reliability into the migration process, ensuring that the resulting application is ready to be hosted on Azure Spring Apps.

Migration

The migration process consists of steps that will help you successfully host a Spring application on Azure Spring Apps. This includes the creation of resources, design considerations, security and performance benchmarking, and, finally, monitoring.

Resources
Design considerations
Security
Performance benchmarking
Monitoring

Here are some of the key considerations for ensuring successful hosting:

- For Spring Cloud apps, Spring Vault secrets can be migrated to Azure Key Vault.
- All certificates can be moved to the Azure Key Vault application; code will need to be configured to access certificates from Key Vault.
- Any explicit configuration server settings in Spring Cloud can be removed. For example, Eureka settings can be removed.
- Azure Spring Apps provides 5 GB of temporary storage per Azure Spring Apps instance, mounted in `/tmp`. Ideally, the application should write data to a persistent store, such as a database or blob storage. In cases where the data exceeds 5 GB, the application code needs to be modified. Once the application is deployed, the output should be directed to the console and not to files.

- Static content for the application can be served from Azure Blob storage; higher-performance data can be served from Azure CDN.

- Any per-service configuration settings need to be added as environment variables.

- Remove any explicit Zipkin dependencies and replace them with Spring Cloud Starters.

- Configure identity provider solutions such as Azure AD to be accessed from the Spring application. A hybrid solution may be needed if the identity provider is on-premises.

- In the case of external on-premises solutions, Azure AD Connect can be used. The application can also use external solutions through SAML/OAuth2 or OpenID.

By following the pre-migration and migration steps, teams can analyze their existing applications and make changes before the application is migrated. Once the application is prepared for migration, resources can be created on Azure as per the finalized design.

Post-migration

Post-migration steps involve validating the functionality and behavior of the application. This includes best practices and guidance on application management.

In post-migration, we will look at some key components, such as security, monitoring the application, and modifications required for Spring components:

- **Monitoring:** Azure Application Insights can monitor the performance and availability of your applications. Azure Monitor alerts can help identify and detect issues with the application quickly.

- **Security:** Use Azure WAF to monitor incoming traffic and Azure Application Gateway for SSL offloading.

- **Availability:** In the case of multi-region deployments, Azure Traffic Manager can help distribute traffic across regions and introduce higher reliability and fault tolerance in the architecture.

- **Spring components:** The application should be designed to work with Spring Cloud Registry. This allows the application to be dynamically discovered by other deployed Spring applications in the infrastructure. This way, the client can find an instance that works if one of the instances is down. All public applications can be accessed via a single endpoint if a Spring Cloud Gateway instance is used.

Post-migration activities target some of the most critical parts of an application. Once the application migration is completed, the post-migration activities take over. This includes factors such as monitoring and security posture. Post-migration typically targets the minimum set of actions that a team must take before an application can be declared as successfully migrated.

A more comprehensive list of post-migration actions is typically the same as the one for best practices for Spring applications.

Section 3

Hosting and deployment options for Azure Spring Apps

There are a few different hosting and deployment options available for Azure Spring Apps , and it’s important to make the right choices for your app and your organization. The majority of Spring apps currently run on an on-premises server or a self-managed cloud server.

Later in this section, we’ll discuss how and why an organization might move an on-premises or self-managed Spring app to a managed PaaS solution.

Public and private applications

Applications can be broadly classified into two categories—public applications, which are accessible to everyone on the internet, and private applications, also known as intranet applications.

Private intranet applications are accessible only to certain employees within the organization. There can also be private applications that are accessible on the web, although only verified users or requests are allowed to gain access to them through identity and authorization solutions.

All Spring applications will fall into one of these categories, so each category has an associated design element that enables or blocks outside requests.

Hosting requirements

Both public and private applications have different needs when it comes to scalability, security, and governance. This requires critical decisions to be made at the time of architecture planning and designing.

Planning address space for the applications is one such requirement in design planning. A well-architected application can effectively manage its IP address requirements. Azure Spring Apps requires two dedicated subnets:

1. Service runtime

2. Spring Boot applications

Each of these subnets requires a dedicated Azure Spring Apps cluster. Multiple clusters cannot share the same subnets. The minimum size of each subnet is `/28`. The number of application instances that Azure Spring Apps can support varies based on the size of the subnet.

Public applications

In public applications, the key changes required are from the network side since a public application needs to be accessible by anyone via the internet. Anyone with the application URL can access these applications. This depends on your business use case.

For example, an e-commerce site is designed to be accessible to anyone with the URL for the site. This means traffic could originate from the internet or other applications hosted on an on-premises network.

All traffic should land on the application gateway, which has a firewall configured with rules determined by the network security team. On-premises applications from your on-premises network should be able to hit the app gateway by utilizing systems such as an ExpressRoute circuit.

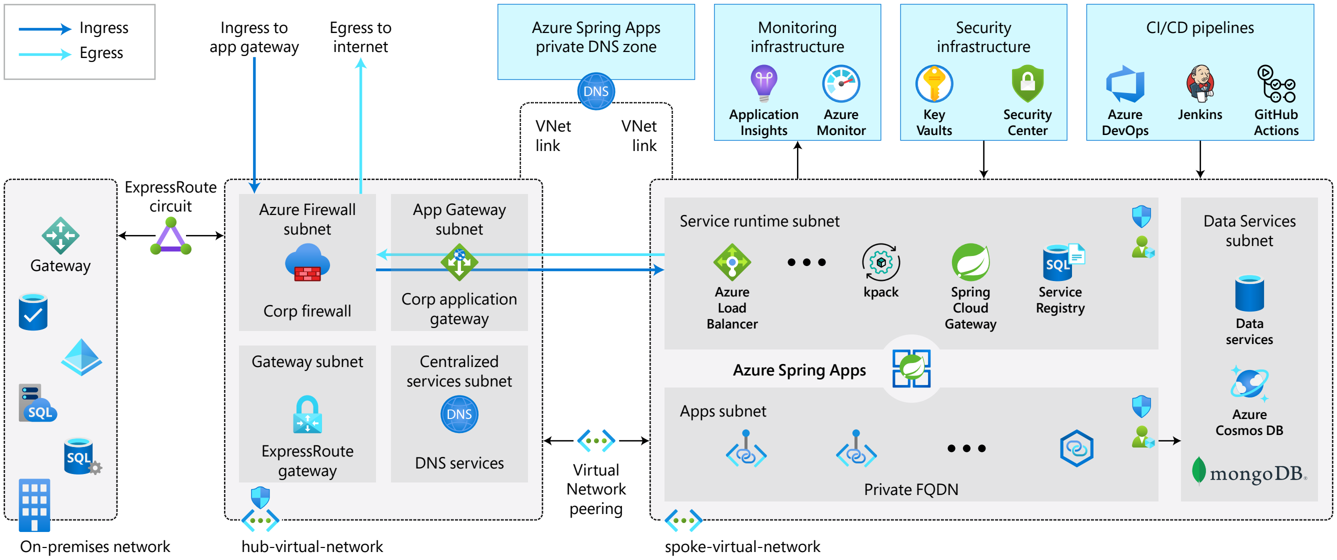


Figure 7: Best practices for a public-facing app on Azure Spring Apps

The architecture in *Figure 7* shows a hub and spoke methodology with connectivity to an on-premises network. All data coming into and going out of the hub passes through the firewall. The hub has a VNET peering with the spoke VNET.

These security and network key points apply to public and private applications:

Security

- Data at rest and in transit should always be encrypted. Similarly, all egress traffic should travel through a firewall, while Application Gateway or Front Door should manage the incoming ingress traffic.

Network

- The internet-routable addresses should be stored in an Azure public DNS, while application host DNS records should be stored in an Azure private DNS. A subnet must only have one instance of Azure Spring Apps.

- Service dependencies should communicate via service endpoints or a private link.

- Name resolution of hosts on-premises and in the cloud should be bidirectional.

Private applications

The architecture for private applications is similar to public applications except for differences in network security rules and security. In private applications, if Azure Spring Apps Config Server is used to load configuration properties from a repository, the repository must be private. All the other points remain the same, and the reference architecture also remains the same in both cases.

Once the nature of the application, whether public or private, has been determined, teams can explore other aspects of the application. One such aspect is the deployment mechanism for the applications. The team can perform manual deployments; however, they are slow and often prone to errors. The recommended approach is to automate the deployments through DevOps.

Deployment options

Using DevOps is a standard practice for automating deployments in your infrastructure. Automating deployment is faster and inherently superior to the manual process. Automating deployments eliminates the possibility of manual errors during deployments, thereby improving the availability of the application. In the following section, we will look at some approaches using popular tools, such as Azure DevOps and GitHub.

CI/CD with Azure DevOps/GitHub

CI/CD is an approach to DevOps that enables teams to deploy applications with ongoing automation and continuous monitoring. This reduces the time required for deployment and eliminates the chance of manual error. The manual approach to deployment can be complex and prone to errors; thus, using an automated approach through DevOps should be the first choice when deploying Azure Spring Apps.

As a starting point, the team should create an Azure Spring Apps instance and have their code ready to be deployed from their DevOps tool of choice. We'll now cover some high-level strategies and steps required for the team to perform their first deployment to Azure Spring Apps using Azure DevOps.

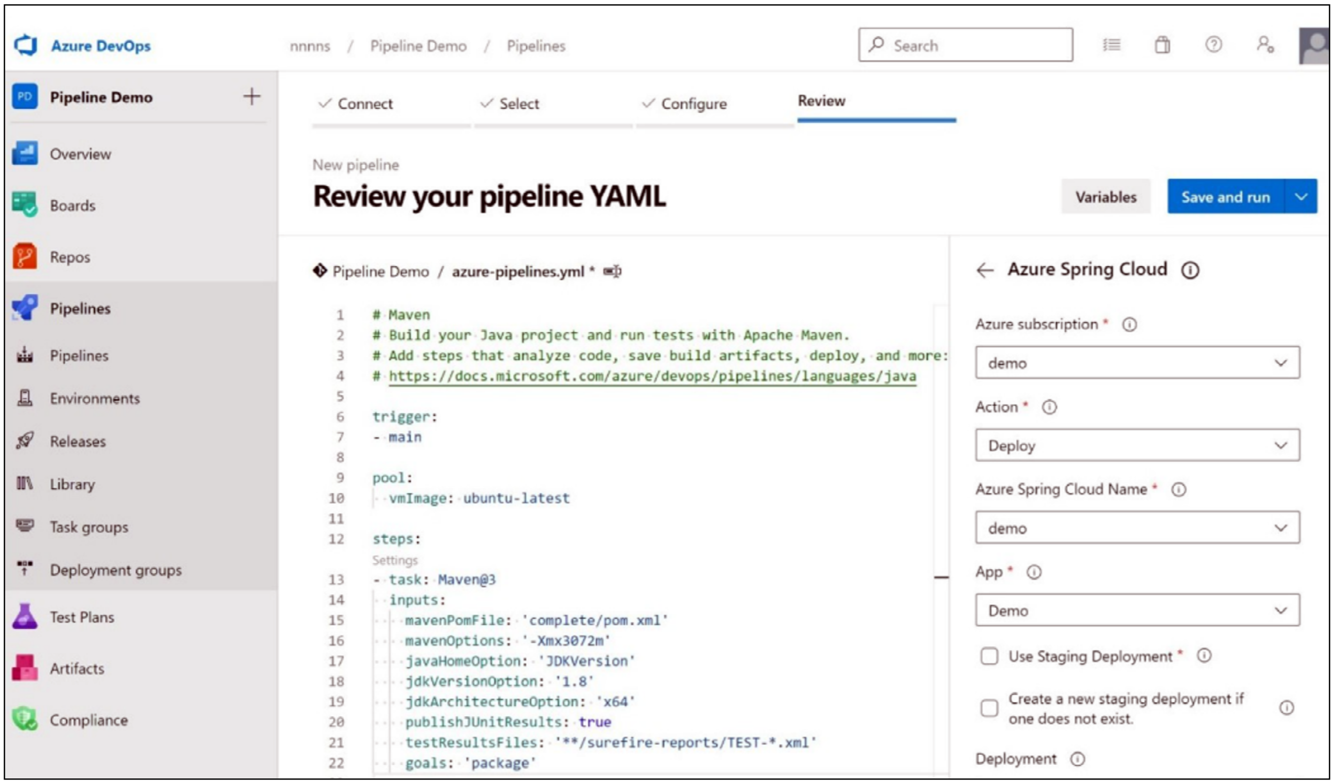
Typically, there are two common types of deployment strategies: rolling and Blue-Green. Both strategies are extremely popular and can be chosen carefully after a review of your business strategy and use cases.

Rolling deployment

In rolling deployment, the strategy is gradually introducing new versions of the application to the environment. Rolling deployment is often the simplest and cheapest strategy to execute and host.

The team should create an ARM service connection to their Azure DevOps project to set up rolling deployments. You can find the instructions here; it should be in the same subscription as your Azure Spring Apps service instance.

First, create a new pipeline in Azure DevOps and use the Azure Spring Apps template. Then, disable "Use Staging Deployment" and set the package or folder value. You can add the following tasks to your pipeline, as shown in *Figure 8*.



New pipeline

Review your pipeline YAML

Pipeline Demo / azure-pipelines.yml *

```yaml
# Maven
# Build your Java project and run tests with Apache Maven.
# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/java

trigger:
- main

pool:
  vmImage: ubuntu-latest

steps:
- task: Maven@3
  inputs:
    mavenPomFile: 'complete/pom.xml'
    mavenOptions: '-Xmx3072m'
    javaHomeOption: 'JDKVersion'
    jdkVersionOption: '1.8'
    jdkArchitectureOption: 'x64'
    publishJUnitResults: true
    testResultsFiles: '**/surefire-reports/TEST-*.xml'
    goals: 'package'
```

Azure Spring Cloud

Azure subscription *: demo

Action *: Deploy

Azure Spring Cloud Name *: demo

App *: Demo

☐ Use Staging Deployment *

☐ Create a new staging deployment if one does not exist.

Deployment

Figure 8: Adding tasks to your pipeline

In the deployment shown in *Figure 8*, the steps in the YAML ensure that the deployment will receive application traffic immediately as soon as it is deployed. This enables testing of the application in the production environment before it gets any external traffic.

Blue-Green deployment

In a Blue-Green model, the team creates two environments at the time of deployment. Both environments are similar in nature. Blue is the current environment, while Green is used to signify the environment with a newer version.

The idea behind Blue-Green deployment is to switch traffic back to an earlier environment if the new version has any technical issues. It improves the reliability posture of the deployments while shielding users from any downtime.

The steps to create a Blue-Green deployment are similar to the preceding method, with the key difference being the creation of another identical environment. Therefore, a main, as well as staging, environment can be created within Azure DevOps under release.

The artifacts created by each of the versions should be made available to the release pipeline. The app’s current release status can be checked by selecting “View release.”

For a detailed set of steps to build release pipelines on Azure DevOps, you can refer to the documentation here. To automate deployments using GitHub Actions, you can refer to the official Microsoft documentation for using GitHub YAML samples for Azure Spring Apps.

Disaster recovery

A Spring application can be deployed to any region that supports the Azure Spring Apps service. For disaster recovery (DR) architectures, the application should be deployed in region pairs. In region pairs, only one region is updated at a time.

This means the application can continue to serve traffic from a different region even when there is a region-level outage. Azure ensures that one region in a region pair is prioritized for recovery.

Any DR architecture should look at three parameters in Azure whenever designing an application:

1. **Azure region pairs:** Microsoft recommends using paired regions within your chosen geographic area to ensure that only one region is updated at a time. Even in the event of a multi-region failure, one of them is prioritized for recovery.

2. **Service availability:** Decide whether your paired regions should run hot/hot, hot/warm, or hot/cold.

3. **Region availability:** Use the geographic area closest to the users. This helps reduce latency and improves response time.

Azure Traffic Manager is a DNS-based traffic load-balancer that can distribute network traffic across multiple regions. Traffic Manager can help direct customers to their closest Azure Spring Apps service instance. This improves the availability and performance of the application.

Typically, in multi-region deployments, a recommended practice is to direct all application traffic through Azure Traffic Manager before sending it to the Azure Spring Apps service.

Azure Traffic Manager is an effective service to control the flow of traffic to your applications in different regions. The Azure Traffic Manager endpoint for each service uses the service IP. Customers should connect to an Azure Traffic Manager DNS name pointing to the Azure Spring Apps service.

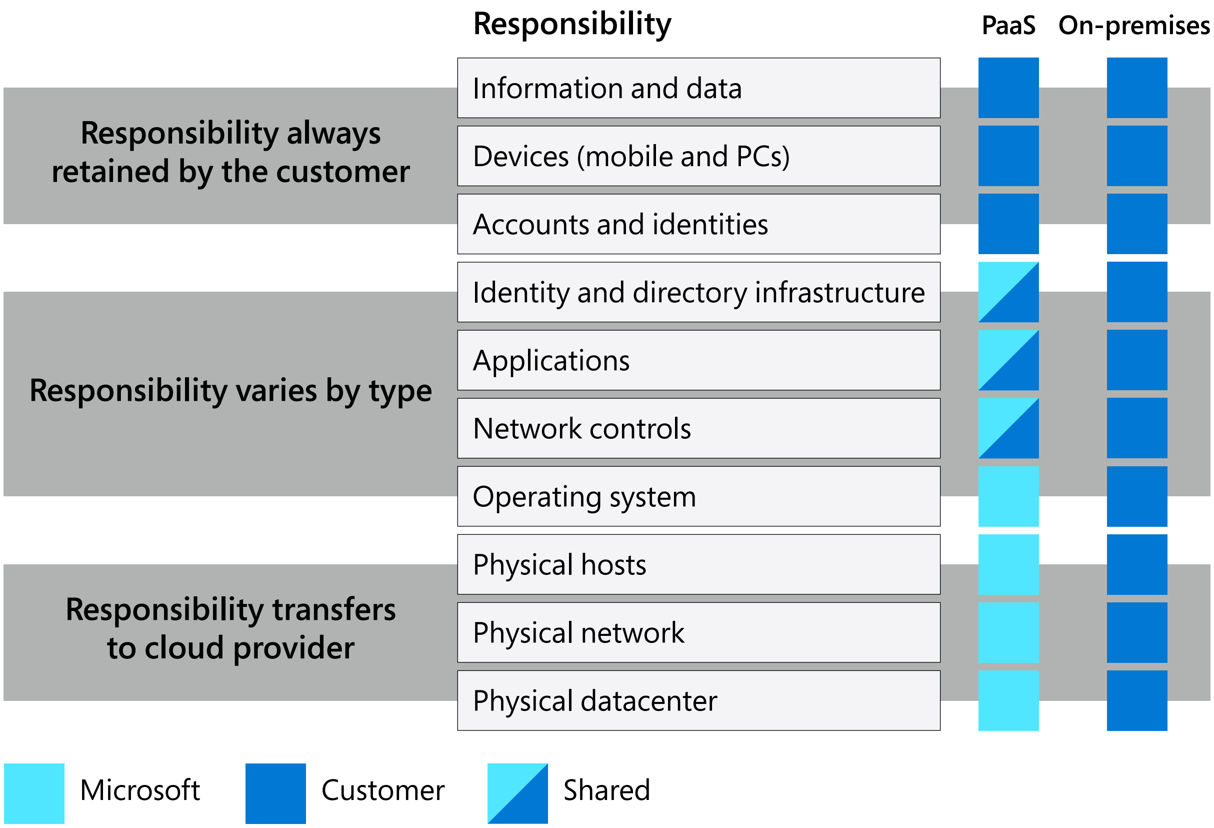
Azure Traffic Manager load-balances traffic across the defined endpoints. If a disaster strikes a datacenter, Azure Traffic Manager will direct traffic from that region to its pair, ensuring service continuity.

These deployment options and disaster recovery considerations need to be taken into account when you've built a new application or when you move an existing application from an on-premises server or self-managed cloud server to a PaaS model.

Moving from self-managed Spring Apps to PaaS

The vast majority of Spring apps today are hosted on either on-premises servers or self-managed servers on the cloud. As the size of IT infrastructure grows, teams look to spend more on performance and optimizing their operations rather than trying to keep applications up and running.

Teams that focus on agility and speed of execution are looking to move away from this model. Given these business shifts, there is a great opportunity for self-managed Spring applications to benefit from modernizing to fully managed solutions on the cloud.



	Responsibility	PaaS	On-premises
Responsibility always retained by the customer	Information and data	■	■
	Devices (mobile and PCs)	■	■
	Accounts and identities	■	■
Responsibility varies by type	Identity and directory infrastructure	◩	■
	Applications	◩	■
	Network controls	◩	■
	Operating system	□	■
Responsibility transfers to cloud provider	Physical hosts	□	■
	Physical network	□	■
	Physical datacenter	□	■

□ Microsoft ■ Customer ◩ Shared

Figure 9: Shared responsibility in the cloud

As the application moves from on-premises or infrastructure as a service (IaaS) to PaaS, the responsibility of managing the underlying infrastructure moves from your team to the cloud provider.

Server management

The Azure cloud offers the capability to manage the infrastructure of the Spring application completely. This empowers the development team by eliminating tasks such as upkeep and management of the server inventory. The Azure cloud manages the latest operating system patches and prevents security vulnerabilities from making their way into the application. Azure ensures that your platform is always patched to the latest patch level, and you only pay for the computing power your application uses. This helps reduce the team's overall spending on hosting and managing the application.

The cost saving, as well as the valuable time saved in managing the application, enables the engineering team to focus on innovation.

Application security

Security is one of the most critical aspects of application design. Azure uses Zero Trust fundamentals to secure an application in the cloud. Most technology teams spend a huge portion of their time and budget on making the applications more secure. Security needs to be baked into the application at each layer rather than being implemented as an afterthought.

Azure ensures the security of the underlying infrastructure for the application, which involves major bug fixes for the operating system that arrive from time to time required to mitigate and prevent security vulnerabilities in the application.

A Microsoft RBAC solution such as Azure AD can be easily plugged into the application to grant roles and permissions to users. Finally, Azure can monitor and filter traffic coming into the application using Azure WAF while continuously testing the security posture using Azure Defender.

Monitoring the application

Azure comes with out-of-the-box monitoring solutions such as Azure Monitor that can be combined with a log aggregating system such as Log Analytics to create a powerful monitoring and aggregation system.

This means application teams can take advantage of the powerful capabilities of Azure Monitor and quickly identify the root cause of issues. Since teams don't need to host a physical or virtual server for monitoring, server administration tasks are removed from the team's responsibilities.

Azure Monitor is easy to set up and can start quickly, offering details on the health of your applications. Application Insights, which is covered in great depth in the next section, can track and trace the health of individual microservices.

All the logs generated can be pushed into a Log Analytics workspace, which can be tracked through Kusto queries. Log Analytics can search a large amount of log data for any insights or events, such as queries that take an unusually long time to process. The teams can deploy the application on Azure and use the already-available monitoring and logging features without requiring a separate dedicated monitoring tool.

Monitoring Azure Spring Apps is essential whether you've moved a self-managed Spring app to a PaaS or you've modernized an old app with a completely fresh approach using Azure Spring Apps.

Section 4

Monitoring Azure Spring Apps

Monitoring and observability are important aspects of any application. Engineering teams need visibility of the performance of an app running in production. This includes some key metrics, such as availability, total requests, and logs. Modern PaaS services, such as Azure Spring Apps, also offer new tools to facilitate the analysis of this data.

When Azure Spring Apps produces logs, it allows the team to add a diagnostic setting to the logs so they can be sent to a retrieval system or an analyzing system, such as Azure Log Analytics. Azure Log Analytics offers the capability of reading or searching the logs to identify or trace an event in the application.

This can be used to also locate any error messages for troubleshooting. Similarly, the logs can be sent to Azure Event Hubs, which can be used to ingest and process events.

Azure Application Insights is an application performance management (APM) tool through which we can trace any event/action or any performance anomaly in the application in real time. Application Insights can provide information on the following areas:

Application map

Performance

Failures

Metrics

Live metrics

Availability

Logs

The screenshot of Application Insights in *Figure 10* shows the interactions between different microservices and their performance. This makes it easy for any team to identify the potential bottlenecks and breakdown areas visually.

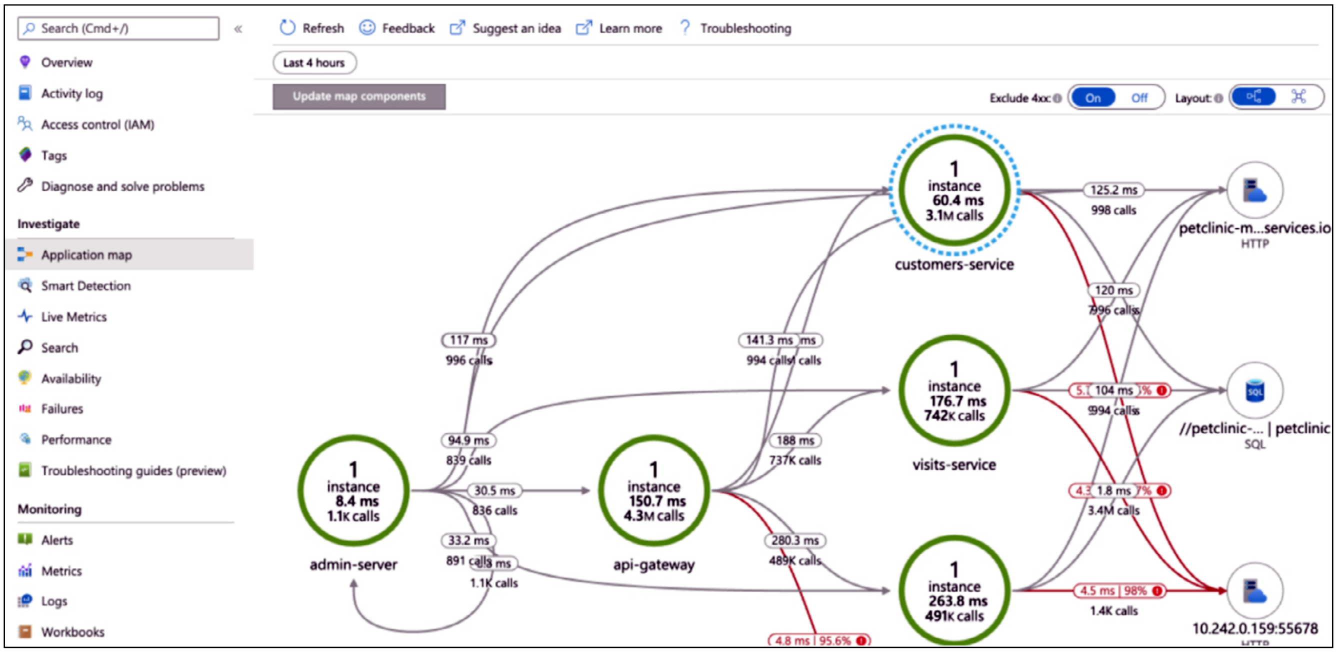
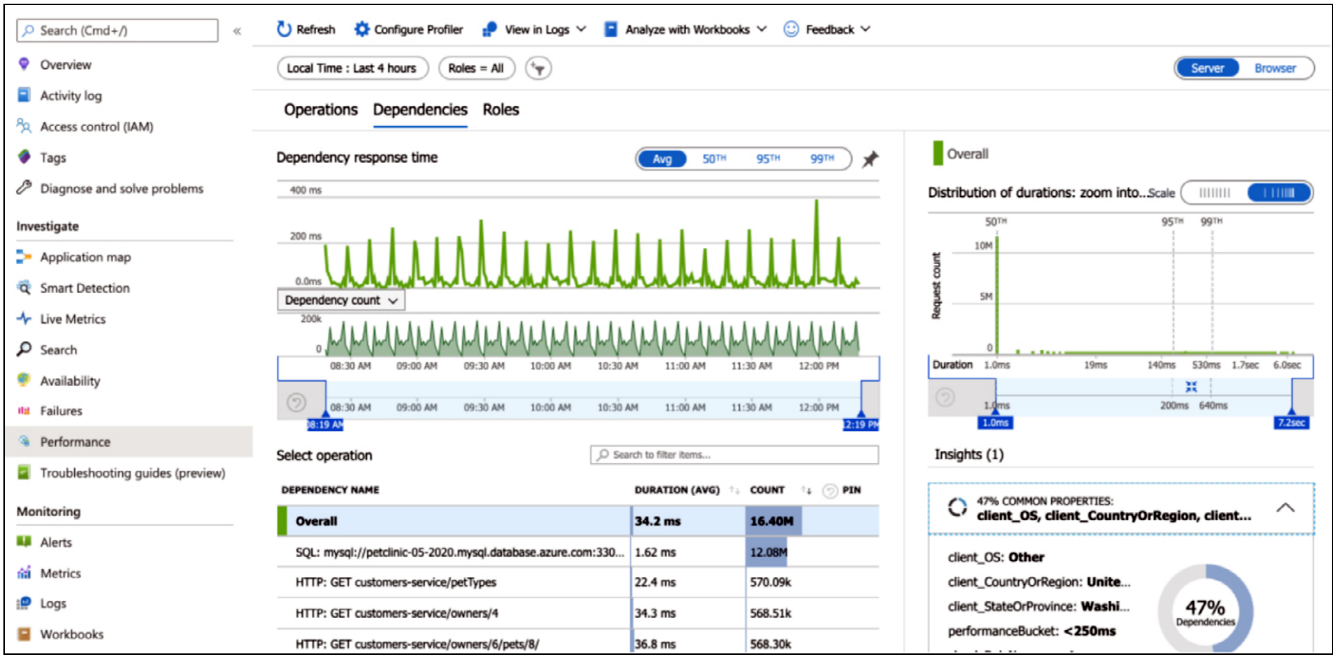


Figure 10: Application map along with performance and live metrics using Application Insights

Application Insights can serve as a single component for all the observability required in the application and is well integrated with other Azure service components. Another area where Application Insights can help is viewing different live metrics related to the application. In *Figure 11*, Application Insights displays live metrics for all individual microservices in an application.



DEPENDENCY NAME	DURATION (AVG)	COUNT	PIN
Overall	34.2 ms	16.40M	
SQL: mysql://petclinic-05-2020.mysql.database.azure.com:330...	1.62 ms	12.08M	
HTTP: GET customers-service/petTypes	22.4 ms	570.09k	
HTTP: GET customers-service/owners/4	34.3 ms	568.51k	
HTTP: GET customers-service/owners/6/pets/8/	36.8 ms	568.30k	

47% COMMON PROPERTIES:
client_OS, client_CountryOrRegion, client...

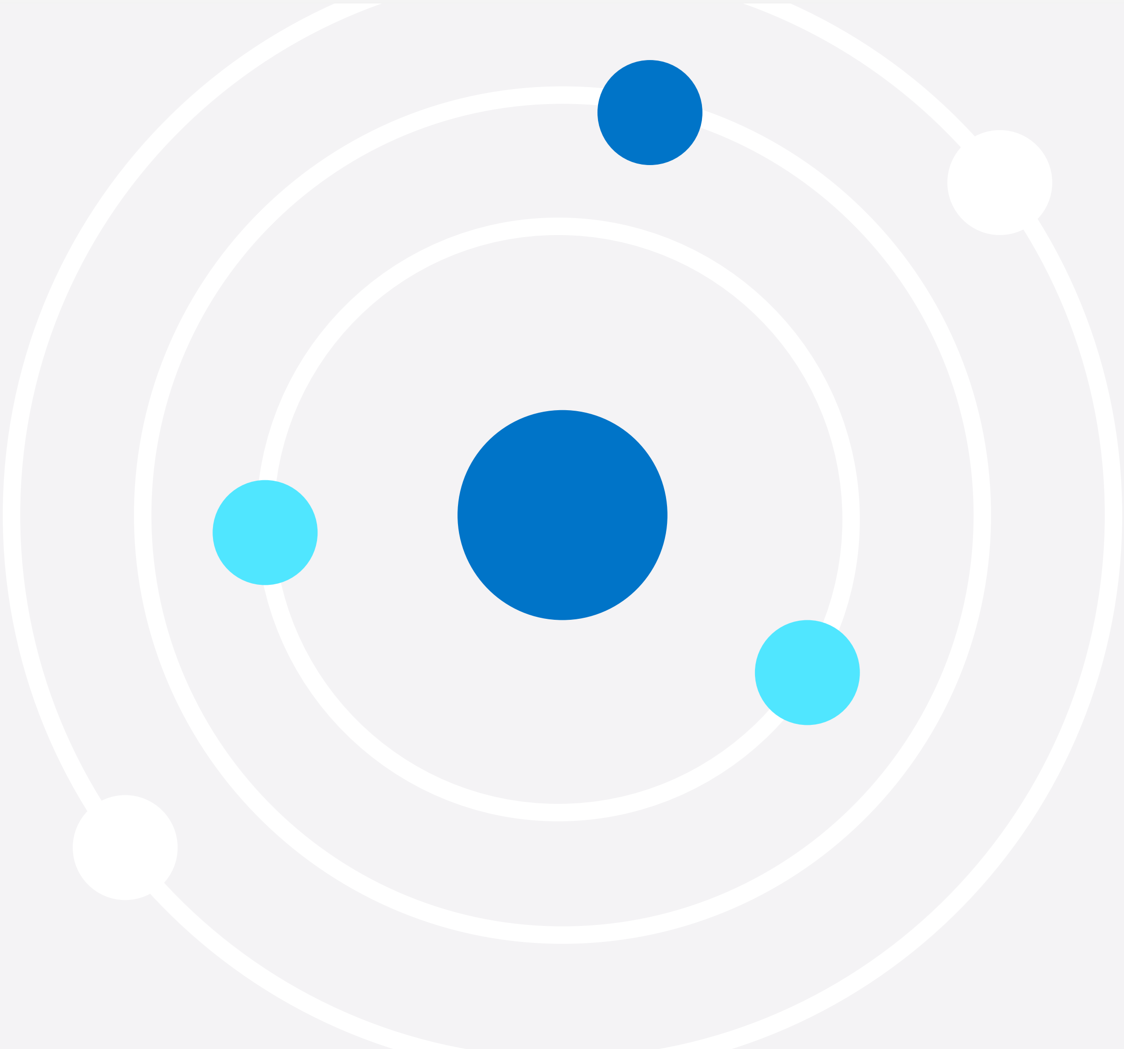
client_OS: **Other**
client_CountryOrRegion: **Unite...**
client_StateOrProvince: **Washi...**
performanceBucket: **<250ms**

Figure 11: Live metrics for individual services

When combined with the application map in *Figure 10*, this view can give deep insights into the performance of any production-grade application. This makes identifying application issues and bottlenecks very easy.

Teams can quickly trace the origin of an individual request and its performance as it traverses through different services. A new feature, "Diagnose and solve problems," can automatically identify some common issues and fix them automatically with no manual help needed by the system administration teams.

Large enterprise applications that might require more feature support can use VMware Tanzu services, which are offered as part of the Enterprise tier in Azure Spring Apps.



Section 5

Choosing the right Azure Spring Apps tier

Azure Spring Apps is offered in three tiers: Basic, Standard, and Enterprise. Each tier is suitable for different use cases. For example, the Azure Spring Apps Basic tier is helpful for development and testing purposes; it is not suitable for production. Therefore, it does not come with a service-level agreement (SLA).

Teams that would like to test their applications on Azure Spring Cloud are most likely to use the Basic tier. Since these are non-production instances, they are smaller in size, and the Azure Spring Apps Basic tier supports a maximum app instance size of 1 vCPU, 2 GB.

Teams looking to host a production-grade application typically opt for the Standard tier. The Azure Spring Apps Standard tier is the most popular tier within Azure Spring Apps. It comes with an SLA and so is suitable for deploying production workloads.

The app instance size is larger in this case and can support a maximum of 500 app instances. The Standard tier of Spring Apps comes with OSS Spring Cloud components. Most general-purpose production workloads find this tier suitable for their use case. However, this tier lacks features such as Spring Cloud Gateway and the API portal; teams who want VMware enterprise features, such as the API portal and gateway, can opt for the Enterprise tier, which offers increased reliability.

The Spring Apps Enterprise tier includes VMware Tanzu technology and offers enterprise-grade support, configurability, flexibility, and portability for enterprise Spring developers. In the Azure Spring Apps Enterprise tier, Tanzu components are enabled on demand according to your needs.

You can select the components you need before creating the service instance. Some of the most common Tanzu features are Tanzu Build Service, Tanzu Application Configuration Service, and Tanzu Service Registry. Tanzu Build Service supports customizable buildpack configurations that automate container creation and governance.

Azure Spring Apps Enterprise includes VMware Spring Runtime support for Spring projects, giving you access to Spring experts to help you make the most of the entire Spring ecosystem.

In the following table, we've outlined some of the major feature differences between the three tiers:

Feature	Basic	Standard	Enterprise
Use case	Dev/Test	Production	Production
Maximum allowed app instance size	1 vCPU, 2 GB	4 vCPUs, 8 GB	4 vCPUs, 8 GB
Maximum allowed app instances	25	500	500
Deploy from source	NA	Open-source buildpacks	Tanzu Buildpacks
Build Service	NA	Available	Available
API portal	NA	NA	Available
Spring Cloud Gateway	NA	NA	Available

Choosing the right tier depends on your use case. Development and test use cases are usually well served by the Basic tier, while a general-level production use case can be covered using the Standard tier. Applications that might look to utilize VMware components for Spring, such as the Spring Cloud Gateway and the API portal, can specifically look to deploy their applications on the Enterprise tier.

Conclusion

Modernizing your apps isn’t just a good idea; it’s essential to keeping your business relevant and functional in a changing technological landscape.

Azure Spring Apps is the platform of choice for modernizing your apps. The service focuses on delivering a great experience to developers with capabilities such as native out-of-the-box integration with Azure services.

The platform manages the entire lifecycle of Spring applications, including monitoring, configuration, and DevOps. This makes it easier for the team to build Java applications at the same time, reducing the overall period from project development to release.

Modernization aims to reduce the time your teams spend on repetitive maintenance and administration tasks. With more time to spend on value, teams experience faster time to market, increasing their overall efficiency. Fully managed PaaS solutions, such as Azure Spring Apps, are the key to unlocking the true value of development teams.

Customer success stories

The following four success stories show how Azure Spring Apps has helped customers make meaningful progress in their modernization journey. Read through them to understand how Azure Spring Apps could help you.

> “‘Cloud-native’ must provide tangible results. Azure Spring Apps helps by taking away the implementation and management effort so we can focus on our core competencies.”
>
> **Jonathan Jones, Lead Solutions Architect, Group Finance IT, Swiss Re**
>
> Read the story

> “Azure Spring Apps was a strategic fit for Liantis because we can really focus on building the practice application, while Microsoft provides and secures the application platform. That’s why Azure Spring Apps is a great fit.”
>
> **Kurt Roggen, Infrastructure and Security Architect, Liantis**
>
> Read the story

> “We are Java developers. We are not infrastructure guys. We are not system administrators. With Azure Spring Apps, we don’t have to worry about managing Kubernetes or cluster downtime.”
>
> **Philipp Stussak, Software Architect, Bosch**
>
> Read the story

> “Azure Spring Apps is paramount to our architecture because of its ease of use and the fact that it’s a fully managed offering. Coupled with the REST APIs that we have developed, we have a truly powerful, resilient, and global platform.”
>
> **Devon Yost, Enterprise Architect, Digital Realty**
>
> Read the story

Resources

- Product overview—Azure Spring Apps
- Azure Spring Apps official documentation
- Deploy your first Spring app on Azure
- Azure Spring Apps training

Modernize with Microsoft today

Simplify and accelerate your cloud journey. Move forward confidently with a proven approach, expert help, and cost-effective offers with the Azure Migration and Modernization Program.

Get started with Azure

Ready when you are—set up your Azure free account.

Get in touch with an Azure Sales Specialist today.