Choose the right Azure service for deploying your Java application

Last modified on 11/16/2020 by judubois@microsoft.com

# Introduction

Organizations often run many different Java workloads, from simple applications to highly complex, mission-critical microservice architectures.

An e-commerce website could have a highly scalable, customer-facing store that relies on hundreds of Spring Boot services. The site could also have a monolithic back-office system to handle its catalog, and many smaller batch applications to update its stocks and notify its customers.

Azure provides many ways to run Java, from simple virtual machines to containers, managed services, or serverless functions. Azure offers one or maybe even several solutions that will fit your needs, including solutions for complex e-commerce scenarios.

This module introduces those solutions, along with tools to help you choose the most appropriate one for your specific use case.

## Learning objectives

By the end of this module, you'll be able to:

* Describe and differentiate the Azure deployment options that are available for Java applications.
* Choose the best Azure service for deploying a Java application.

## Prerequisites

* Familiarity with developing and running Java applications
* Beginner-level experience with cloud infrastructure

# Application deployment types

There are several ways to deploy Java applications to the cloud. This unit explores the various options so that, in the next unit, you can better understand the services that Azure provides.

## Virtual machines, containers, or platform as a service?

The main question is whether you want or need to deploy your application on a virtual machine (VM), inside a container, or as a platform as a service (PaaS) solution.

* With **virtual machines**, you're in a world similar to an on-premises or classical datacenter environment. Azure provides a set of preconfigured VMs that run the main operating systems, Windows and Linux, and you'll need to configure and maintain those machines.

We suggest that you adopt this solution initially, because it's the closest to what most enterprises are already using before they move to the cloud. They usually install their own configuration management software, install their favorite version of Java, and can then run their Java workload in a way that's similar how they've done it in the past.

The VM solution works well if you have an experienced operations team that will configure and maintain them, and if you have specific use cases. For example, you might be using some native libraries or some proprietary software, such as Oracle WebLogic Server or IBM WebSphere Application Server.

* With **containers**, you still have most of the control you have with VMs, but with less operations effort. You can install your own Java Virtual Machine (JVM), or some specific software, and your containers will run either locally or on any cloud provider.

Because containers offer a lot of freedom, they suffer from some of the same issues as VMs. If you provide your own JVM, you'll need to update and patch it, as necessary. As a result, Docker images require a good continuous integration and continuous delivery (CI/CD) toolchain to be able to maintain containers properly. Because Docker images can run locally and are lighter than VMs, they also provide a great developer experience.

* With a **platform as a service** solution, most of the maintenance and operation burden is handled by the cloud provider. Operating system (OS) updates, Java patches, security, and compliance are all provided. As a result, this option is usually more secure and less costly. It also comes with some scalability features, which should allow your application to adapt better to your customers' needs. It also results in better performance under load and lower cost when there is less traffic.

You can achieve more by using a PaaS solution. You can set up automatic configuration, manage and load secrets (for example, by using Azure Key Vault), monitor your application, launch a live profiling session, and enable zero-downtime deployment.

## Deployment options

Whether you use VMs, containers, or a platform as a service solution, you can usually deploy your Java applications to the cloud in either of two ways:

* **Source code deployment**: You commit your source code to a Git repository, and the cloud provider runs a process that compiles, builds, and packages the application.
* **JAR, WAR, or EAR file deployment**: You package your application, usually as an executable JAR (Java ARchive) file, but WAR (Web Application ARchive), EAR (Enterprise Application ARchive), and other file formats are also possible. The executable file is then run by the cloud provider.

These two deployment options are classical ways to run Java applications. For both options, the build process is usually similar, and the main difference is where that process is run. Letting the cloud provider do the build is simpler and, with this approach, the cloud provider applies its own security checks and patches. By building the application locally or by using a CI/CD platform, such as GitHub Actions, you get more flexibility and control.

## Serverless functions

Serverless functions or, more specifically, Azure Functions, is a mix of various solutions we've seen and offers a very specific feature: serverless functions are meant to run for short periods of time. Usually, a function is awakened by an event, such as an HTTP request, and it stays "hot" for a few minutes until it goes back to sleep.

Functions share features with the platform as a service solution we've described earlier. In Azure, our PaaS solution (Azure App Service) and our serverless solution (Azure Functions) are technically similar and share some common code and services.

For deployment options, functions usually work with JAR files. Other options such as Docker are available, but they're less popular and usually don't perform as well. This is because the underlying platform can't optimize them in the same way that it can for JAR files.

By their nature, serverless functions need to be specifically coded. Their features depend on the cloud provider they run on, and their short life makes it complicated to use traditional solutions, such as caching or HTTP session replication.

Serverless functions can scale well, and they offer the best price for low-usage environments. At the same time, they can respond to the most demanding traffic loads.

# Compare the solutions

We've described the most typical deployment types for Java applications. Let's now see which Azure service can work best, depending on the deployment type you need.

## Azure App Service

Azure App Service is a platform as a service (PaaS) solution provided by Azure. It offers an easy-to-use service for running Java web applications, with automatic OS and Java Virtual Machine (JVM) updates and patches, as well as autoscaling or monitoring.

App Service can run any executable JAR file, such as the files provided by Spring Boot, and it can deploy any WAR application on top of popular application servers such as Apache Tomcat or Red Hat JBoss.

For running a simple Java monolithic application, App Service would therefore be the best option.

If you need more control over your deployments, App Service also supports Docker images. This benefit makes it possible for you to include specific or custom solutions, but at the cost of more maintenance at your end.

## Azure Functions

Azure Functions is the serverless platform provided by Azure, and it supports running Java workloads.

Running a Java application on Azure Functions usually requires some minor configuration and setup. By their nature, serverless functions have short lives (usually, only a few minutes), so some of the services that you would traditionally use might not be as efficient or practical as Azure App Service.

As a result, Azure Functions is tailored for applications that need a lot of scalability. You can stop them regularly to reduce cost, and you can scale them up to respond to high loads.

## Azure Spring Cloud

Azure Spring Cloud is a unique Azure service that provides a platform as a service solution for running Spring Boot microservices. Spring Cloud provides many tools, such as a service registry or a configuration server. It also provides specific autoscaling and security mechanisms that greatly simplify the use of a microservice architecture that's based on Spring Boot.

The Spring Cloud offering is created and managed jointly by Microsoft and VMware, the company that builds the Spring Framework. Spring Cloud provides support both for OS and JVM updates and patches, such as Azure App Service, and for advanced Spring services.

You can also use the Azure Spring Cloud service to run other workloads, such as Spring Boot monolithic applications or even .NET microservices, which use Steeltoe, but the service is mainly used to run Spring Boot microservices.

## Azure Kubernetes Service

Azure Kubernetes Service (AKS) is a managed and secured Kubernetes solution provided by Azure. It can run any Docker image, and it benefits from belonging to the whole Kubernetes ecosystem. AKS works well if you want to deliver cloud-native applications in any language, including Java.

Azure Kubernetes Service is a great solution if you have heterogenous workloads or if you want precise control over your environment. AKS runs Java workloads well, but you'll need to build and maintain your Docker images.

## Azure VMs

Running VMs on Azure gives you the greatest flexibility and variety of options. You can choose between multiple operating systems, typically Windows, and the main Linux distributions. You can bring your own software, or use some of the provided images.

For Java specifically, you mainly use VMs to:

* Install and run a Java web application manually on top of Windows or Linux.
* Install and run Docker or even AKS to run an image that contains a Java web application.
* Use one of the supported offers, such as Oracle WebLogic, to run Java web applications on your VMs.

When you use Azure VMs, there's no limit to what you can install and run, because you have full control over them. The main drawback with VMs, though, is that you'll need to maintain and secure them. For VMs, this maintenance work is your responsibility.

# Choose the right destination

We've discussed the available options for deploying a Java application in the cloud and, more specifically, on Azure. But which option best addresses your specific needs? Here's a guide to help you choose.

## Platform as a service or infrastructure control?

The first question to ask is whether you want full control over your application or you want to use a platform as a service (PaaS) solution. Your answer usually depends on the needs of the team you're working with.

Platform as a service solutions are easier to use. Many teams prefer them because they offer lower cost, better reliability, and greater security.

Teams choose PaaS solutions for the following reasons:

* They need to go to production quickly, and they have only a limited operations team or no ops team at all.
* They have certain high availability, security, or compliance requirements, and these needs can be met by out-of-box features that are provided by a managed service.
* They might not have the resources or the willingness to maintain a production-grade infrastructure.

But some teams will prefer to have a full control over their infrastructure, mostly for the following reasons:

* If you're using specific proprietary software or noncommon Java features, a managed service might not be the right fit for you. For example, you might want to use network broadcasting to find your application nodes, and this feature isn't available on a managed service.
* You might want to use a specific native library, which also won't be able to run on a managed service.

## Choose a PaaS solution on Azure

For running Java workloads on a platform as a service solution, Azure provides three possible options:

* **Azure Functions**: Best for simple event-driven workloads that can scale extremely well.
* **Azure App Service**: Best for normal web applications or for running Jakarta EE workloads.
* **Azure Spring Cloud**: Best for running Spring Boot applications and, more specifically, Spring Cloud microservices.

## Choose infrastructure control on Azure

For full control over your deployments, Azure provides the following two main options:

* **Azure Kubernetes Service (AKS)**: Best for running Docker images, You also benefit from the rich AKS ecosystem.
* **Azure virtual machines**: Best for running any kind of workload, or if you want to "lift and shift" from an existing on-premises infrastructure without rearchitecting your applications.

## Final words to help you decide

Deciding on a destination is only a first step. As you're choosing your service, keep in mind the following:

* Your maintenance tasks will vary greatly depending on both the type of service you choose and the skill set of your own operations team.
* Your developers will use the platform daily, and they'll be more productive if the service provides a great developer experience.

For both of these points, your choice depends on the team you have. The best practice is to choose a destination that matches your team's expertise and preferences.

# Knowledge check

## Check your knowledge

### Why would you use a platform as a service (PaaS) solution?

* ⬜ You want to go to production as fast as possible, with a limited ops team.
  + You are partially right, but this answer is not the only reason.
* ⬜ You have high availability, security, or compliance requirements.
  + You are partially right, but this answer is not the only reason.
* ⬜ Your team doesn't have the resources or the willingness to maintain a production-grade infrastructure.
  + You are partially right, but this answer is not the only reason.
* ❎All of the above.
  + Correct. Platform as a service solutions are the preferred choice if you want to go to production quickly, with high confidence and a minimal maintenance burden.

### Which Azure service is designed to run serverless workloads?

* ⬜ Azure App Service
  + Incorrect. Azure App Service is a platform as a service solution, which can run standard Java applications.
* ❎Azure Functions
  + Correct. Azure Functions is the serverless offer from Azure.
* ⬜ Azure Kubernetes Service
  + Incorrect. Azure Kubernetes Service is a managed Kubernetes offer
* ⬜ Azure Spring Cloud
  + Incorrect. Azure Spring Cloud is a managed service for running Spring Boot and Spring Cloud applications.

### With a platform as a service offer such as Azure App Service or Azure Spring Cloud, who manages your Java Virtual Machine (JVM)?

* ⬜ You are responsible for updating and patching your JVM.
  + Incorrect. Microsoft manages, updates, and patches your JVM for you.
* ❎Microsoft is responsible for updating and patching your JVM.
  + Correct. Microsoft manages, updates, and patches your JVM for you.
* ⬜ JVM can't be updated or patched.
  + Incorrect. Microsoft manages, updates, and patches your JVM for you.

# Summary

In this module, you learned about the main deployment types that are used for Java applications. We discussed platform as a service (PaaS) offers, virtual machines (VMs), Azure Kubernetes Service (AKS), and serverless solutions. You also learned how to select the Azure service that best addresses your team's needs: Azure App Service, Azure Functions, Azure Spring Cloud, Azure Kubernetes Service, or Azure Virtual Machines.

## Additional resources

For more information about the Azure services described in this module, see:

* [Azure App Service](https://azure.microsoft.com/services/app-service/)
* [Azure Functions](https://azure.microsoft.com/services/functions/)
* [Azure Spring Cloud](https://azure.microsoft.com/services/spring-cloud/)
* [Azure Kubernetes Service](https://azure.microsoft.com/services/kubernetes-service/)
* [Azure Virtual Machines](https://azure.microsoft.com/services/virtual-machines/)