Introduction

**Learning objectives**

After completing this module, you'll be able to:

* Describe Azure App Service key components and value.
* Explain how Azure App Service manages authentication and authorization.
* Identify methods to control inbound and outbound traffic to your web app.
* Deploy an app to App Service using Azure CLI commands.

Examine Azure App Service

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favourite programming language or framework. Applications run and scale with ease on both Windows and Linux-based environments.

Built-in auto scale support

The ability to scale up/down or scale out/in is baked into the Azure App Service. Depending on the usage of the web app, you can scale the resources of the underlying machine that is hosting your web app up/down. Resources include the number of cores or the amount of RAM available. Scaling out/in is the ability to increase, or decrease, the number of machine instances that are running your web app.

Container support

With Azure App Service, you can deploy and run containerized web apps on Windows and Linux. You can pull container images from a private Azure Container Registry or Docker Hub. Azure App Service also supports multi-container apps, Windows containers, and Docker Compose for orchestrating container instances.

Continuous integration/deployment support

The Azure portal provides out-of-the-box continuous integration and deployment with Azure DevOps Services, GitHub, Bitbucket, FTP, or a local Git repository on your development machine. Connect your web app with any of the above sources and App Service will do the rest for you by auto-syncing code and any future changes on the code into the web app. Continuous integration and deployment for containerized web apps is also supported using either Azure Container Registry or Docker Hub.

Deployment slots

When deploying a web app you can use a separate deployment slot instead of the default production slot when you're running in the Standard App Service Plan tier or better. Deployment slots are live apps with their own host names. App content and configurations elements can be swapped between two deployment slots, including the production slot.

App Service on Linux

App Service can also host web apps natively on Linux for supported application stacks. It can also run custom Linux containers (also known as Web App for Containers). App Service on Linux supports many language specific built-in images. Just deploy your code. Supported languages and frameworks include: Node.js, Java (JRE 8 & JRE 11), PHP, Python, .NET, and Ruby. If the runtime your application requires isn't supported in the built-in images, you can deploy it with a custom container.

The languages, and their supported versions, are updated regularly. You can retrieve the current list by using the following command in the Cloud Shell.

az webapp list-runtimes --os-type linux

Limitations

App Service on Linux does have some limitations:

* App Service on Linux isn't supported on Shared pricing tier.
* The Azure portal shows only features that currently work for Linux apps. As features are enabled, they're activated on the portal.
* When deployed to built-in images, your code and content are allocated a storage volume for web content, backed by Azure Storage. The disk latency of this volume is higher and more variable than the latency of the container filesystem. Apps that require heavy read-only access to content files might benefit from the custom container option, which places files in the container filesystem instead of on the content volume.

Examine Azure App Service plans

In App Service, an app always runs in an App Service plan. An App Service plan defines a set of compute resources for a web app to run. One or more apps can be configured to run on the same computing resources (or in the same App Service plan).

When you create an App Service plan in a certain region (for example, West Europe), a set of compute resources is created for that plan in that region. Whatever apps you put into this App Service plan run on these compute resources as defined by your App

Service plan. Each App Service plan defines:

1. Operating System (Windows, Linux)
2. Region (West US, East US, etc.)
3. Number of VM instances
4. Size of VM instances (Small, Medium, Large)
5. Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated, IsolatedV2)

The pricing tier of an App Service plan determines what App Service features you get and how much you pay for the plan. There are a few categories of pricing tiers:

1. **Shared compute**: Free and Shared, the two base tiers, runs an app on the same Azure VM as other App Service apps, including apps of other customers. These tiers allocate CPU quotas to each app that runs on the shared resources, and the resources can't scale out.
2. **Dedicated compute**: The Basic, Standard, Premium, PremiumV2, and PremiumV3 tiers run apps on dedicated Azure VMs. Only apps in the same App Service plan share the same compute resources. The higher the tier, the more VM instances are available to you for scale-out.
3. **Isolated**: The Isolated and IsolatedV2 tiers run dedicated Azure VMs on dedicated Azure Virtual Networks. It provides network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities.

**Note**

App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure virtual machines as other App Service apps. Some apps might belong to other customers. These tiers are intended to be used only for development and testing purposes.

How does my app run and scale?

In the Free and Shared tiers, an app receives CPU minutes on a shared VM instance and can't scale out. In other tiers, an app runs and scales as follows:

* An app runs on all the VM instances configured in the App Service plan.
* If multiple apps are in the same App Service plan, they all share the same VM instances.
* If you have multiple deployment slots for an app, all deployment slots also run on the same VM instances.
* If you enable diagnostic logs, perform backups, or run WebJobs, they also use CPU cycles and memory on these VM instances.

In this way, the App Service plan is the scale unit of the App Service apps. If the plan is configured to run five VM instances, then all apps in the plan run on all five instances. If the plan is configured for autoscaling, then all apps in the plan are scaled out together based on the autoscale settings.

What if my app needs more capabilities or features?

Your App Service plan can be scaled up and down at any time. It's as simple as changing the pricing tier of the plan. If your app is in the same App Service plan with other apps, you may want to improve the app's performance by isolating the compute resources. You can do it by moving the app into a separate App Service plan.

You can potentially save money by putting multiple apps into one App Service plan. However, since apps in the same App Service plan all share the same compute resources you need to understand the capacity of the existing App Service plan and the expected load for the new app.

Isolate your app into a new App Service plan when:

* The app is resource-intensive.
* You want to scale the app independently from the other apps in the existing plan.
* The app needs resource in a different geographical region.

This way you can allocate a new set of resources for your app and gain greater control of your apps.

Deploy to App Service

Every development team has unique requirements that can make implementing an efficient deployment pipeline difficult on any cloud service. App Service supports both automated and manual deployment.

Automated deployment

Automated deployment, or continuous deployment, is a process used to push out new features and bug fixes in a fast and repetitive pattern with minimal effect on end users.

Azure supports automated deployment directly from several sources. The following options are available:

1. **Azure DevOps Services**: You can push your code to Azure DevOps Services, build your code in the cloud, run the tests, generate a release from the code, and finally, push your code to an Azure Web App.
2. **GitHub**: Azure supports automated deployment directly from GitHub. When you connect your GitHub repository to Azure for automated deployment, any changes you push to your production branch on GitHub are automatically deployed for you.
3. **Bitbucket**: With its similarities to GitHub, you can configure an automated deployment with Bitbucket.

Manual deployment

There are a few options that you can use to manually push your code to Azure:

1. **Git**: App Service web apps feature a Git URL that you can add as a remote repository. Pushing to the remote repository deploys your app.
2. **CLI**: webapp up is a feature of the az command-line interface that packages your app and deploys it. Unlike other deployment methods, az webapp up can create a new App Service web app for you if you haven't already created one.
3. **Zip deploy**: Use curl or a similar HTTP utility to send a ZIP of your application files to App Service.
4. **FTP/S**: FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.

Use deployment slots

Whenever possible, use deployment slots when deploying a new production build. When using a Standard App Service Plan tier or better, you can deploy your app to a staging environment and then swap your staging and production slots. The swap operation warms up the necessary worker instances to match your production scale, thus eliminating downtime.

Continuously deploy code

If your project has designated branches for testing, QA, and staging, then each of those branches should be continuously deployed to a staging slot. This allows your stakeholders to easily assess and test the deployed branch.

Continuously deploy containers

For custom containers from Azure Container Registry or other container registries, deploy the image into a staging slot and swap into production to prevent downtime. The automation is more complex than code deployment because you must push the image to a container registry and update the image tag on the webapp.

1. **Build and tag the image**: As part of the build pipeline, tag the image with the git commit ID, timestamp, or other identifiable information. It’s best not to use the default “latest” tag. Otherwise, it’s difficult to trace back what code is currently deployed, which makes debugging far more difficult.
2. **Push the tagged image**: Once the image is built and tagged, the pipeline pushes the image to our container registry. In the next step, the deployment slot will pull the tagged image from the container registry.
3. **Update the deployment slot with the new image tag**: When this property is updated, the site will automatically restart and pull the new container image.