100 XP

# Introduction

3 minutes

Your company is considering moving their web apps to the cloud and has asked you to evaluate Azure App Service.

## 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.

## Next unit: Examine Azure App Service

Continue

# Examine Azure App Service

3 minutes

Azure App Service is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favorite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and Linux-based environments.

## Built-in auto scale support

Baked into Azure App Service is the ability to scale up/down or scale out/in. Depending on the usage of the web app, you can scale your app up/down the resources of the underlying machine that is hosting your web app. 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.

## Continuous integration/deployment support

The Azure portal provides out-of-the-box continuous integration and deployment with Azure DevOps, 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.

## Deployment slots

Using the Azure portal, or command-line tools, you can easily add deployment slots to an App Service web app. For instance, you can create a staging deployment slot where you can push your code to test on Azure. Once you are happy with your code, you can easily swap the staging deployment slot with the production slot. You do all this with a few simple mouse clicks in the Azure portal.

> **Note**
>
> Deployment slots are only available in the Standard, Premium, and Isolated plan tiers.

# 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 a number of language specific built-in images. Just deploy your code. Supported languages include: Node.js, Java (JRE 8 & JRE 11), PHP, Python, .NET Core, and Ruby. If the runtime your application requires is not supported in the built-in images, you can deploy it with a custom container.

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

Bash Copy

```
az webapp list-runtimes --os-type linux
```

## Limitations

App Service on Linux does have some limitations:

- App Service on Linux is not supported on Shared pricing tier.
- You can't mix Windows and Linux apps in the same App Service plan.
- Historically, you could not mix Windows and Linux apps in the same resource group. However, all resource groups created on or after January 21, 2021 do support this scenario. Support for resource groups created before January 21, 2021 will be rolled out across Azure regions (including National cloud regions) soon.
- The Azure portal shows only features that currently work for Linux apps. As features are enabled, they're activated on the portal.

---

## Next unit: Examine Azure App Service plans

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Examine Azure App Service plans

3 minutes

In App Service, an app (Web Apps, API Apps, or Mobile Apps) 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). In addition, Azure Functions also has the option of running in an 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:

- Region (West US, East US, etc.)
- Number of VM instances
- Size of VM instances (Small, Medium, Large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, PremiumV2, PremiumV3, Isolated)

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:

- **Shared compute**: Both **Free** and **Shared** share the resource pools of your apps with the 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.
- **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.
- **Isolated**: This tier runs 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.
- **Consumption:** This tier is only available to *function apps*. It scales the functions dynamically depending on workload.

> **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 is 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.

## Next unit: Deploy to App Service

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Deploy to App Service

3 minutes

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 integration, is a process used to push out new features and bug fixes in a fast and repetitive pattern with minimal impact on end users.

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

- **Azure DevOps**: You can push your code to Azure DevOps, 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.
- **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 will be automatically deployed for you.
- **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:

- **Git**: App Service web apps feature a Git URL that you can add as a remote repository. Pushing to the remote repository will deploy your app.
- **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.

- **Zip deploy**: Use `curl` or a similar HTTP utility to send a ZIP of your application files to App Service.
- **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.

---

## Next unit: Explore authentication and authorization in App Service

Continue >

---

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Explore authentication and authorization in App Service

3 minutes

Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code in your web app, API, and mobile back end, and also Azure Functions.

## Why use the built-in authentication?

You're not required to use App Service for authentication and authorization. Many web frameworks are bundled with security features, and you can use them if you like. If you need more flexibility than App Service provides, you can also write your own utilities.

The built-in authentication feature for App Service and Azure Functions can save you time and effort by providing out-of-the-box authentication with federated identity providers, allowing you to focus on the rest of your application.

- Azure App Service allows you to integrate a variety of auth capabilities into your web app or API without implementing them yourself.
- It's built directly into the platform and doesn't require any particular language, SDK, security expertise, or even any code to utilize.
- You can integrate with multiple login providers. For example, Azure AD, Facebook, Google, Twitter.

## Identity providers

App Service uses federated identity, in which a third-party identity provider manages the user identities and authentication flow for you. The following identity providers are available by default:

| Provider | Sign-in endpoint | How-To guidance |
|---|---|---|
| Microsoft Identity Platform | `/.auth/login/aad` | App Service Microsoft Identity Platform login |
| Facebook | `/.auth/login/facebook` | App Service Facebook login |
| Google | `/.auth/login/google` | App Service Google login |
| Twitter | `/.auth/login/twitter` | App Service Twitter login |
| Any OpenID Connect provider | `/.auth/login/<providerName>` | App Service OpenID Connect login |

When you enable authentication and authorization with one of these providers, its sign-in endpoint is available for user authentication and for validation of authentication tokens from the provider. You can provide your users with any number of these sign-in options.

# How it works

The authentication and authorization module runs in the same sandbox as your application code. When it's enabled, every incoming HTTP request passes through it before being handled by your application code. This module handles several things for your app:

- Authenticates users with the specified provider
- Validates, stores, and refreshes tokens
- Manages the authenticated session
- Injects identity information into request headers

The module runs separately from your application code and is configured using app settings. No SDKs, specific languages, or changes to your application code are required.

> **Note**
>
> In Linux and containers the authentication and authorization module runs in a separate container, isolated from your application code. Because it does not run in-process, no direct integration with specific language frameworks is possible.

## Authentication flow

The authentication flow is the same for all providers, but differs depending on whether you want to sign in with the provider's SDK.

- Without provider SDK: The application delegates federated sign-in to App Service. This is typically the case with browser apps, which can present the provider's login page to the user. The server code manages the sign-in process, so it is also called *server-directed flow* or *server flow*.
- With provider SDK: The application signs users in to the provider manually and then submits the authentication token to App Service for validation. This is typically the case with browser-less apps, which can't present the provider's sign-in page to the user. The application code manages the sign-in process, so it is also called *client-directed flow* or *client flow*. This applies to REST APIs, Azure Functions, JavaScript browser clients, and native mobile apps that sign users in using the provider's SDK.

The table below shows the steps of the authentication flow.

| Step | Without provider SDK | With provider SDK |
|---|---|---|
| Sign user in | Redirects client to `/.auth/login/<provider>`. | Client code signs user in directly with provider's SDK and receives an authentication token. For information, see the provider's documentation. |
| Post-authentication | Provider redirects client to `/.auth/login/<provider>/callback`. | Client code posts token from provider to `/.auth/login/<provider>` for validation. |
| Establish authenticated session | App Service adds authenticated cookie to response. | App Service returns its own authentication token to client code. |
| Serve authenticated content | Client includes authentication cookie in subsequent requests (automatically handled by browser). | Client code presents authentication token in `X-ZUMO-AUTH` header (automatically handled by Mobile Apps client SDKs). |

For client browsers, App Service can automatically direct all unauthenticated users to `/.auth/login/<provider>`. You can also present users with one or more `/.auth/login/<provider>`

links to sign in to your app using their provider of choice.

## Authorization behavior

In the Azure portal, you can configure App Service with a number of behaviors when an incoming request is not authenticated.

- **Allow unauthenticated requests:** This option defers authorization of unauthenticated traffic to your application code. For authenticated requests, App Service also passes along authentication information in the HTTP headers.This option provides more flexibility in handling anonymous requests. It lets you present multiple sign-in providers to your users.
- **Require authentication:** This option will reject any unauthenticated traffic to your application. This rejection can be a redirect action to one of the configured identity providers. In these cases, a browser client is redirected to `/.auth/login/<provider>` for the provider you choose. If the anonymous request comes from a native mobile app, the returned response is an `HTTP 401 Unauthorized`. You can also configure the rejection to be an `HTTP 401 Unauthorized` or `HTTP 403 Forbidden` for all requests.

> ⊗ **Caution**
>
> Restricting access in this way applies to all calls to your app, which may not be desirable for apps wanting a publicly available home page, as in many single-page applications.

## Next unit: Discover App Service networking features

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Discover App Service networking features

3 minutes

By default, apps hosted in App Service are accessible directly through the internet and can reach only internet-hosted endpoints. But for many applications, you need to control the inbound and outbound network traffic.

There are two main deployment types for Azure App Service. The multitenant public service hosts App Service plans in the Free, Shared, Basic, Standard, Premium, PremiumV2, and PremiumV3 pricing SKUs. There is also the single-tenant App Service Environment (ASE) hosts Isolated SKU App Service plans directly in your Azure virtual network.

## Multi-tenant App Service networking features

Azure App Service is a distributed system. The roles that handle incoming HTTP or HTTPS requests are called *front ends*. The roles that host the customer workload are called *workers*. All the roles in an App Service deployment exist in a multi-tenant network. Because there are many different customers in the same App Service scale unit, you can't connect the App Service network directly to your network.

Instead of connecting the networks, you need features to handle the various aspects of application communication. The features that handle requests to your app can't be used to solve problems when you're making calls from your app. Likewise, the features that solve problems for calls from your app can't be used to solve problems to your app.

| Inbound features | Outbound features |
| --- | --- |
| App-assigned address | Hybrid Connections |
| Access restrictions | Gateway-required virtual network integration |
| Service endpoints | Virtual network integration |

| Inbound features | Outbound features |
| --- | --- |
| Private endpoints | |

You can mix the features to solve your problems with a few exceptions. The following inbound use cases are examples of how to use App Service networking features to control traffic inbound to your app.

| Inbound use case | Feature |
| --- | --- |
| Support IP-based SSL needs for your app | App-assigned address |
| Support unshared dedicated inbound address for your app | App-assigned address |
| Restrict access to your app from a set of well-defined addresses | Access restrictions |

# Default networking behavior

Azure App Service scale units support many customers in each deployment. The Free and Shared SKU plans host customer workloads on multitenant workers. The Basic and higher plans host customer workloads that are dedicated to only one App Service plan. If you have a Standard App Service plan, all the apps in that plan will run on the same worker. If you scale out the worker, all the apps in that App Service plan will be replicated on a new worker for each instance in your App Service plan.

## Outbound addresses

The worker VMs are broken down in large part by the App Service plans. The Free, Shared, Basic, Standard, and Premium plans all use the same worker VM type. The PremiumV2 plan uses another VM type. PremiumV3 uses yet another VM type. When you change the VM family, you get a different set of outbound addresses. If you scale from Standard to PremiumV2, your outbound addresses will change. If you scale from PremiumV2 to PremiumV3, your outbound addresses will change. In some older scale units, both the inbound and outbound addresses will change when you scale from Standard to PremiumV2.

There are a number of addresses that are used for outbound calls. The outbound addresses used by your app for making outbound calls are listed in the properties for your app. These addresses are shared by all the apps running on the same worker VM family in the App Service deployment. If you want to see all the addresses that your app might use in a scale unit, there's a property called `possibleOutboundAddresses` that will list them.

## Find outbound IPs

To find the outbound IP addresses currently used by your app in the Azure portal, click **Properties** in your app's left-hand navigation.

You can find the same information by running the following command in the Cloud Shell. They are listed in the **Additional Outbound IP Addresses** field.

Bash Copy

```
az webapp show \
    --resource-group <group_name> \
    --name <app_name> \
    --query outboundIpAddresses \
    --output tsv
```

To find all possible outbound IP addresses for your app, regardless of pricing tiers, run the following command in the Cloud Shell.

Bash Copy

```
az webapp show \
    --resource-group <group_name> \
    --name <app_name> \
    --query possibleOutboundIpAddresses \
    --output tsv
```

## Next unit: Exercise: Create a static HTML web app by using Azure Cloud Shell

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Exercise: Create a static HTML web app by using Azure Cloud Shell

15 minutes

This module requires a sandbox to complete.

A **sandbox** gives you access to free resources. Your personal subscription will not be charged. The sandbox may only be used to complete training on Microsoft Learn. Use for any other reason is prohibited, and may result in permanent loss of access to the sandbox.

Microsoft provides this lab experience and related content for educational purposes. All presented information is owned by Microsoft and intended solely for learning about the covered products and services in this Microsoft Learn module.

Activate sandbox

The free sandbox allows you to create resources in a subset of the Azure global regions. Select a region from the following list when you create resources:

- West US 2
- South Central US
- Central US
- East US
- West Europe
- Southeast Asia
- Japan East
- Brazil South
- Australia Southeast
- Central India

In this exercise, you'll deploy a basic HTML+CSS site to Azure App Service by using the Azure CLI `az webapp up` command. You'll then update the code and redeploy it by using the same command.

The `az webapp up` command makes it easy to create and update web apps. When executed it performs the following actions:

- Create a default resource group if one isn't specified.
- Create a default app service plan.
- Create an app with the specified name.
- Zip deploy files from the current working directory to the web app.

# Download the sample app

In this section you'll use the sandbox to download the sample app and set variables to make some of the commands easier to enter.

1. In the sandbox create a directory and then navigate to it.

```
Bash
Copy

mkdir htmlapp

cd htmlapp
```

2. Run the following `git` command to clone the sample app repository to your *htmlapp* directory.

```
Bash
Copy

git clone https://github.com/Azure-Samples/html-docs-hello-world.git
```

3. Set variables to hold the resource group and app names by running the following commands.

```
Bash
Copy

resourceGroup=$(az group list --query "[].{id:name}" -o tsv)
appName=az204app$RANDOM
```

# Create the web app

1. Change to the directory that contains the sample code and run the `az webapp up` command. In the following example, replace `<myLocation>` with a region near you.

```bash
cd html-docs-hello-world

az webapp up -g $resourceGroup -n $appName --html
```

This command may take a few minutes to run. While running, it displays information similar to the example below.

```json
{
"app_url": "https://<myAppName>.azurewebsites.net",
"location": "westeurope",
"name": "<app_name>",
"os": "Windows",
"resourcegroup": "<resource_group_name>",
"serverfarm": "appsvc_asp_Windows_westeurope",
"sku": "FREE",
"src_path": "/home/<username>/demoHTML/html-docs-hello-world ",
< JSON data removed for brevity. >
}
```

2. Open a new tab in your browser and navigate to the app URL (`https://<myAppName>.azurewebsites.net`) and verify the app is running - take note of the title at the top of the page. Leave the browser open on the app for the next section.

> **Note**
>
> You can copy `<myAppName>.azurewebsites.net` from the output of the previous command, or select the URL in the output to open the site in a new tab.

# Update and redeploy the app

1. In the Cloud Shell, type `code index.html` to open the editor. In the `<h1>` heading tag, change *Azure App Service - Sample Static HTML Site* to *Azure App Service Updated* - or to anything else that you'd like.

2. Use the commands **ctrl-s** to save and **ctrl-q** to exit.

3. Redeploy the app with the same `az webapp up` command you used earlier.

```
Bash
az webapp up -g $resourceGroup -n $appName --html
```

> **Tip**
>
> You can use the up arrow on your keyboard to scroll through previous commands.

4. Once deployment is completed switch back to the browser from step 2 in the "Create the web app" section above and refresh the page.

## Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

# Knowledge check

3 minutes

## Check your knowledge

**1.** Which of the following App Service plans supports only function apps?

- ○ Dedicated
- ○ Isolated
- ◉ Consumption

✔ That's correct. The consumption tier is only available to function apps. It scales the functions dynamically depending on workload.

**2.** Which of the following networking features of App Service can be used to control outbound network traffic?

- ○ App-assigned address
- ◉ Hybrid Connections

✔ That's correct. Hybrid Connections are an outbound network feature.

- ○ Service endpoints

## Next unit: Summary

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

# Summary

3 minutes

In this module, you learned how 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.

## Module complete:

Continue to next module

How are we doing? ☆ ☆ ☆ ☆ ☆