Deploy a website to Azure with Azure App Service

Host a web application with Azure App service

What is Azure App Service?

Azure App Service is a fully managed web application hosting platform. This platform as a service (PaaS) offered by Azure allows you to focus on designing and building your app while Azure takes care of the infrastructure to run and scale your applications.

Deployment slots

Using the Azure portal, 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.

Creating a web app

Field	***Description***
Subscription	A valid and active Azure subscription.
Resource group	A valid resource group.
App name	The name of the web app. This name becomes part of the app's URL, so it must be unique among all Azure App Service web apps.
Publish	You can deploy your application to App Service as code or as a ready-to-run Docker image. Selecting Docker image will activate the Docker tab of the wizard, where you provide information about the Docker registry from which App Service will retrieve your image.
Runtime stack	If you choose to deploy your application as code, App Service needs to know what runtime your application uses (examples include Node.js, Python, Java, and .NET). If you deploy your application as a Docker image, you will not need to choose a runtime stack, since your image will include it.

Operating system	App Service can host applications on Windows or Linux servers. See below for additional information.
Region	The Azure region from which your application will be served.
App Service Plan	See below for information about App Service plans.

Operating systems

If you are deploying your app as code, many of the available runtime stacks are limited to one operating system or the other. After choosing a runtime stack, the toggle will indicate whether or not you have a choice of operating system. If your target runtime stack is available on both operating systems, select the one that you use to develop and test your application.

If your application is packaged as a Docker image, choose the operating system on which your image is designed to run.

Selecting **Windows** activates the Monitoring tab, where you have the option to enable **Application Insights**. Enabling this feature will configure your app to automatically send detailed performance telemetry to the Application Insights monitoring service without requiring any changes to your code. Application Insights can be used from Linux-hosted apps as well, but this turnkey, no-code option is only available on Windows.

App Service plans

An **App Service** plan is a set of virtual server resources that run App Service apps. A plan's **size** (sometimes referred to as its **sku** or **pricing tier**) determines the performance characteristics of the virtual servers that run the apps assigned to the plan and the App Service features that those apps have access to. Every App Service web app you create must be assigned to a single App Service plan that runs it.

A single App Service plan can host multiple App Service web apps. In most cases, the number of apps you can run on a single plan will be limited by the performance characteristics of the apps and the resource limitations of the plan.

App Service plans are the unit of billing for App Service. The size of each App Service plan in your subscription, in addition to the bandwidth resources used by the apps deployed to those plans, determines the price that you pay. The number of web apps deployed to your App Service plans has no effect on your bill.

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 (previously known as Visual Studio Team 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.
- **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.
- **OneDrive**: Microsoft's cloud-based storage. You must have a Microsoft Account linked to a OneDrive account to deploy to Azure.
- **Dropbox**: Azure supports deployment from Dropbox, which is a popular cloud-based storage system that is similar to OneDrive.

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.
- **az webapp up**: 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.
- **Zipdeploy**: Use az webapp deployment source config-zip to send a ZIP of your application files to App Service. Zipdeploy can also be accessed via basic HTTP utilities such as curl.
- **Visual Studio**: Visual Studio features an App Service deployment wizard that can walk you through the deployment process.
- **FTP/S**: FTP or FTPS is a traditional way of pushing your code to many hosting environments, including App Service.

Links

- App Service on Azure Stack overview
- Deploy to Azure App Service using Visual Studio Code
- Configure deployment sources for App Services on Azure Stack
- Set up staging environments in Azure App Service

- Deployment FAQs for Web Apps in Azure

Summary

1. True or false: Azure App service can automatically scale your web application to meet traffic demand?

True

Azure App service has built-in auto scale support and will increase or decrease the resources allocated to run your app as needed, depending on the demand.

2. Which of the following is not a valid automated deployment source?

SharePoint

Azure currently supports Azure DevOps, GitHub, Bitbucket, OneDrive, Dropbox, and external Git repositories.

Publish a web app to Azure with Visual Studio

What is the Azure App Service?

Azure App Service is a service for hosting web applications, REST APIs, and backend services. App Service supports code written in .NET Core, .NET Framework, Java, Ruby, Node.js, PHP, and Python. App Service is ideal for most websites, particularly if you don't need tight control over the hosting infrastructure.

What is the App Service plan?

The App Service plan defines the compute resources your app will consume, where those resources are located, how many additional resources the plan can consume, and the pricing tier. These compute resources are analogous to the server farm in conventional web hosting. One or more apps can be configured to run on the same App Service plan.

When you deploy your apps, you can create an App Service plan or you can continue to add apps to an existing plan. However, apps in the same App Service plan share the same compute resources. To determine whether the new app has the necessary resources, you need to understand the capacity of the existing App Service plan, and the expected load for the new app. Overloading an App Service plan can cause reduced performance or downtime for your new and existing apps.

You can define an App Service plan in advance in the Azure portal with PowerShell or the Azure CLI, or set one up as you publish your application in Visual Studio.

Each App Service plan defines:

- Region (West US, East US, and so on)
- Number of VM instances
- Size of VM instances (small, medium, large)
- Pricing tier (Free, Shared, Basic, Standard, Premium, Premium V2, Isolated)

Select a region

When creating an App Service plan, you have to define a region or location where that plan will be hosted. Typically, you would choose a region geographically close to your expected customers.

Pricing and reliability levels

Shared compute: **Free** and **Shared**, the two base tiers, run 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 cannot scale-out.

Free and Shared plans are best for small-scale personal projects with limited traffic demands, with a set limit of 165 MB of outbound data every 24 hours.

Dedicated compute: The **Basic**, **Standard**, **Premium**, and **Premium V2** 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.

The Standard service plan is best suited for live production workloads, where you are publishing commercial applications to customers.

The Premium service plans support high-capacity web apps where you do not want the additional costs of a dedicated (isolated) plan.

Isolated: This tier runs dedicated Azure VMs on dedicated Azure virtual networks, which provide network isolation on top of compute isolation to your apps. It provides the maximum scale-out capabilities. You would only select an Isolated service plan when you have a specific requirement for the highest levels of security and performance.

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 resources in a different geographical region

Your App Service plan can be scaled up and down at any time. You can choose a lower pricing tier at first and scale up later when you need more App Service features.

Specify the resource group

A resource group is a logical container into which Azure resources like web apps, databases, and storage accounts are deployed and managed. It is a mechanism for organizing resources for the purpose of management, monitoring and billing. You can use an existing resource group or create one directly from Visual Studio.

Stage a web app deployment for testing and rollback by using App Service deployment slots

Use a deployment slot

Within a single Azure App Service web app, you can create multiple deployment slots. Each slot is a separate instance of that web app, and it has a separate host name. You can deploy a different version of your web app into each slot.

One slot is the production slot. This slot is the web app that users see when they connect. Make sure that the app deployed to this slot is stable and well tested.

Use additional slots to host new versions of your web app. Against these instances, you can run tests such as integration tests, acceptance tests, and capacity tests. Fix any problems before you move the code to the production slot. Additional slots behave like their own App Service instances, so you can have confidence that your tests show how the app will run in production.

After you're satisfied with the test results for a new app version, deploy it by swapping its slot with the production slot. Unlike a code deployment, a slot swap is instantaneous. When you swap slots, the slot host names are exchanged, immediately sending production traffic to the new version of the app. When you use slot swaps to deploy, your app is never exposed to the public web in a partially deployed state.

If you find that, in spite of your careful testing, the new version has a problem, you can roll back the version by swapping the slots back.

Understand slots as separate Azure resources

When you use more than one deployment slot for a web app, those slots are treated as separate instances of that web app. For example, they're listed separately on the **All resources** page in the Azure portal. They each have their own URL. However, each slot shares the resources of the App Service plan, including virtual machine memory and CPU as well as disk space.

Avoid a cold start during swaps

Many of the technologies that developers use to create web apps require final compilation and other actions on the server before they deliver a page to a user. Many of these tasks are completed when the app starts up and receives a request. For example, if you use ASP.NET to build your app, code is compiled and views are completed when the first user requests a page. Subsequent requests for that page receive a faster response because the code is already compiled.

The initial delay is called a cold start. You can avoid a cold start by using slot swaps to deploy to production. When you swap a slot into production, you "warm up" the app because your action sends a request to the root of the site. The warm-up request ensures that all

compilation and caching tasks finish. After the swap, the site responds as fast as if it had been deployed for days.

Access a slot

The new slot is effectively a separate web app with a different host name. That's why anyone on the internet can access it if they know that host name. Unless you register the slot with a search engine or link to it from a crawled page, the slot won't appear in search engine indexes. It will remain obscure to the general internet user.

You can control access to a slot by using IP address restrictions. Create a list of IP address ranges that you'll allow to access the slot or a list of ranges that you'll deny access to the slot. These lists are like the allow ranges and deny ranges that you can set up on a firewall. Use this list to permit access only to computers that belong to your company or development team.

Manage the configuration for a swap

When you swap two slots, the app's configuration travels to the new slot along with the app. You can override this behavior for individual application settings and configuration strings by configuring them as **slot settings**.

Suppose, for example, you have two databases. You use one for production and the other for acceptance testing. You always want the app version in the staging slot to use the testing database. The app version in the production slot should always use the production database. To achieve this, you can configure the database connection string as a slot setting.

Understand the slot-swapping preview

When you swap slots, the settings in the target slot (which is typically the production slot) are applied to the app version in the source slot before the host names are swapped. You might discover problems at this point. For example, if the database connection string is configured as a slot setting, the new version of the web app will use the existing production database. If you forgot to upgrade the database schema in the production database before the swap, you could see errors and exceptions when the new app version attempts to use the old schema.

To help you discover problems before your app goes live into production, Azure App Service offers a swap-with-preview feature. When you choose this option, the swap proceeds in two phases:

1. **Phase 1**: Slot settings from the target slot are applied to the web app in the source slot. Then Azure warms up the staging slot. At this point, the swap operation pauses so you can test the app in the source slot to make sure it works with the target slot configuration. If you find no problems, begin the next phase.
2. **Phase 2**: The host names for the two sites are swapped. The version of the app now in the source slot receives its slot settings.

Auto swap

Auto swap brings the zero-downtime and easy rollback benefits of swap-based deployment to automated deployment pipelines. When you configure a slot for auto swap, Azure automatically swaps it whenever you push code or content into that slot.

When you use auto swap, you can't test the new app version in the staging slot before the swap. Auto swap mainly benefits users who want zero-downtime deployments and simple automated deployment pipelines.

If you want to be able to test before you swap, you'll need a more complex deployment pipeline that requests the slot swap itself. Alternatively, you can deploy to a separate slot that's dedicated for testing.

Links

Set up staging environments in App Service.

Learn about Azure subscriptions, service limits, quotas, and constraints.

Find out more about App Service static IP restrictions.

Summary

1. How does Azure App Service ensure that production performance doesn't drop just after a swap?
App Service warms up the app by sending a request to the root of the site.
The warm-up request ensures that all compilation and caching tasks finish before the slots are swapped.
2. Which of the following are NOT shared between all of the deployment slots for a web app?
Host names
The slots for a web app have different host names. They share the same virtual machines and deployment plans.

Scale an App Service web app to efficiently meet demand with App Service scale up and scale out

It's important to be able to scale a web app for these reasons:

- It enables the app to remain responsive during periods of high demand.
- It helps to save you money by reducing the resources required when demand drops.

Azure App Service enables you to meet these goals by providing scale up and down, and scale in and out.

App Service plans and scalability

A web app running in Azure typically uses Azure App Service to provide the hosting environment. App Service can arrange for multiple instances of the web app to run and will load balance incoming requests across these instances. Each instance runs on a virtual machine.

The resources available to each instance are defined by an App Service plan. The App Service plan specifies the operating system (Windows or Linux), the hardware (memory, CPU processing capacity, disk storage, and so on), and the availability of services like automatic backup and restore.

Azure provides a series of well-defined App Service plan tiers. This list summarizes each of these tiers, in increasing order of capacity (and cost):

- The Free tier provides 1 GB of disk space and support for up to 10 apps, but only a single shared instance and no SLA for availability. Each app has a compute quota of 60 minutes per day. The Free service plan is mainly suitable for app development and testing rather than production deployments.
- The Shared tier provides support for more apps (up to 100) also running on a single shared instance. Apps have a compute quota of 240 minutes per day. There is no availability SLA.
- The Basic tier supports an unlimited number of apps and provides more disk space. Apps can be scaled out to three dedicated instances. This tier provides an SLA of 99.95% availability. There are three levels in this tier that offer varying amounts of compute power, memory, and disk storage.
- The Standard tier also supports an unlimited number of apps. This tier can scale to 10 dedicated instances and has an availability SLA of 99.95%. Like the Basic tier, this tier has three levels that offer an increasingly powerful set of compute, memory, and disk options.
- The Premium tier gives you up to 20 dedicated instances, an availability SLA of 99.95%, and multiple levels of hardware.
- The Isolated tier runs in a dedicated Azure virtual network, which gives you network and compute isolation. This tier can scale out to 100 instances and has an availability SLA of 99.95%.

Scale up a web app

You scale an App Service plan up and down by changing the pricing tier and hardware level that it runs on. You can start with the Free tier and scale up as needed according to your requirements. This process is manual. You can also scale down again if you no longer need the resources associated with a particular tier.

Scaling up can cause an interruption in service to client apps running at the time. They might need to disconnect from the service and reconnect if the scale-up occurs during an active call to the web app. And new connections might be rejected until scaling finishes. Also, scaling up can cause the outgoing IP addresses for the web app to change. If your web app depends on other services that have firewalls restricting incoming traffic, you'll need to reconfigure these services.

As with scale-out, you should monitor the performance of your system to ensure that scaling up (or down) has the desired effect. It's also important to understand that scale up and scale out can work cooperatively together. If you have scaled out to the maximum number of instances available for your pricing tier, you must scale up before you can scale out further.

Links

Azure App Service plan overview

Scale up an app in Azure

Deploy and run a containerized web app with Azure App Service

What is Container Registry?

Container Registry is an Azure service that you can use to create your own private Docker registries. Like Docker Hub, Container Registry is organized around repositories that contain one or more images. Container Registry also lets you automate tasks such as redeploying an app when an image is rebuilt.

Security is an important reason to choose Container Registry instead of Docker Hub:

- You have much more control over who can see and use your images.
- You can sign images to increase trust and reduce the chances of an image becoming accidentally (or intentionally) corrupted or otherwise infected.
- All images stored in a container registry are encrypted at rest.

Working with images in Container Registry is like working with Docker Hub, but offers a few unique benefits:

- Container Registry runs in Azure. The registry can be replicated to store images near where they're likely to be deployed.
- Container Registry is highly scalable, providing enhanced throughput for Docker pulls that can span many nodes concurrently. The Premium SKU of Container Registry includes 500 GiB of storage.

Deploy a web app from a repository in Azure Container Registry

When you create a web app from a Docker image, you configure the following properties:

- The registry that contains the image. The registry can be Docker Hub, Azure Container Registry, or some other private registry.
- The image. This item is the name of the repository.
- The tag. This item indicates which version of the image to use from the repository. By convention, the most recent version is given the tag latest when it's built.
- Startup File. This item is the name of an executable file or a command to be run when the image is loaded. It's equivalent to the command that you can supply to Docker when running an image from the command line by using docker run. If you're deploying a ready-to-run, containerized app that already has the ENTRYPOINT and/or COMMAND values configured, you don't need to fill this in.

After you've configured the web app, the Docker image is pulled and run as a "cold start" operation the first time a user attempts to visit the site. The app might take a few seconds to start initially, but thereafter it will be available immediately.

What is a webhook?

Azure App Service supports continuous deployment using webhooks. A webhook is a service offered by Azure Container Registry. Services and applications can subscribe to the webhook to receive notifications about updates to images in the registry. A web app that uses App Service can subscribe to an Azure Container Registry webhook to receive notifications about updates to the image that contains the web app. When the image is updated, and App Service receives a notification, your app automatically restarts the site and pulls the latest version of the image.

What is the Container Registry tasks feature?

You use the tasks feature of Container Registry to rebuild your image whenever its source code changes automatically. You configure a Container Registry task to monitor the GitHub repository that contains your code and trigger a build each time it changes. If the build finishes successfully, Container Registry can store the image in the repository. If your web app is set up for continuous integration in App Service, it receives a notification via the webhook and updates the app.

Let's use these two features to enable continuous integration from the App Service.

Links

Container Registry

Use a custom Docker image for Web App for Containers

Tutorial: Build and deploy container images in the cloud with Azure Container Registry Tasks

Tutorial: Automate container image builds in the cloud when you commit source code

List of Azure CLI commands to manage private Azure Container Registries