

Contents

[App Service Documentation](#)

[Overview](#)

[About App Service](#)

[About App Service Environments](#)

[Compare hosting options](#)

[Quickstarts](#)

[ASP.NET](#)

[Node.js](#)

[PHP](#)

[Java](#)

[Python](#)

[Ruby](#)

[Static HTML site](#)

[ARM template](#)

[Custom container](#)

[Multi-container app](#)

[App on Azure Arc \(preview\)](#)

[Tutorials](#)

[Map custom domain](#)

[Secure domain with TLS/SSL](#)

[App with DB](#)

[ASP.NET Core with SQL DB](#)

[ASP.NET with SQL DB](#)

[PHP with MySQL](#)

[Node.js with MongoDB](#)

[Ruby with Postgres](#)

[Python with Postgres](#)

[Java with Spring Data](#)

[Custom containers](#)

- [Deploy from ACR](#)
- [Multi-container app](#)
- [Secretless back end connectivity](#)
- [Access SQL DB with managed identity](#)
- [Access services without managed identity support](#)
- [Authenticate users](#)
- [Secure app accesses storage, and Microsoft Graph](#)
 - [Overview](#)
 - [Set up App Service authentication](#)
 - [Secure app accesses storage](#)
 - [Secure app accesses Microsoft Graph as the user](#)
 - [Secure app accesses Microsoft Graph as the app](#)
 - [Clean up resources](#)
- [Isolate network traffic](#)
- [Host a RESTful API](#)
- [Add CDN](#)
- [Send email/tweet](#)
- [Troubleshoot errors](#)
- [GitHub Actions](#)
 - [ASP.NET Core with a container](#)
 - [ASP.NET](#)
- [Samples](#)
 - [Azure CLI](#)
 - [Azure PowerShell](#)
 - [Resource Manager templates](#)
 - [Terraform](#)
 - [Bicep](#)
- [Concepts](#)
 - [App Service plans](#)
 - [Cost management](#)
 - [OS functionality](#)
 - [Deployment best practices](#)

Azure Arc hosting (preview)

Security

[Recommendations](#)

[Authentication and authorization](#)

[OS and runtime patching](#)

[Security baseline](#)

[Security controls by Azure Policy](#)

[Security overview](#)

Networking

[Overview](#)

[Inbound and outbound IPs](#)

[Private endpoint](#)

[Hybrid connections](#)

[Virtual network integration](#)

[Application Gateway integration](#)

[NAT gateway integration](#)

[Traffic Manager integration](#)

Local cache

Diagnostics

How-To guides

Configure app

[Configure common settings](#)

[Configure ASP.NET](#)

[Configure ASP.NET Core](#)

[Configure Node.js](#)

[Configure PHP](#)

[Configure Python](#)

[Configure Java](#)

[Configure Ruby](#)

[Configure custom container](#)

[Configure Azure Storage \(container\)](#)

Deploy to Azure

Deploy the app

- [Run from package](#)
- [Deploy ZIP or WAR](#)
- [Deploy via FTP](#)
- [Deploy via cloud sync](#)
- [Deploy continuously](#)
- [Custom containers CI/CD](#)
- [Deploy from local Git](#)
- [GitHub Actions](#)
- [GitHub Actions for Containers](#)
- [Deploy with template](#)
- [Set deployment credentials](#)
- [Create staging environments](#)
- [Resource Manager template guidance](#)

Map custom domain

- [Buy domain](#)
- [Map domains with Traffic Manager](#)
- [Migrate an active domain](#)

Authenticate users

- [Configure identity providers](#)
 - [Authenticate with Azure AD](#)
 - [Authenticate with Facebook](#)
 - [Authenticate with Google](#)
 - [Authenticate with Twitter](#)
 - [Authenticate with an OpenID Connect provider \(Preview\)](#)
 - [Authenticate using Sign in with Apple \(Preview\)](#)

Customize sign-ins/outs

Access user identities

Work with tokens

Manage API versions

File-based configuration

Secure app

- [Add TSL/SSL cert](#)
- [Restrict access](#)
- [Use a managed identity](#)
- [Reference secrets from Key Vault](#)
- [Use TLS/SSL cert in code](#)
- [Configure TLS mutual authentication](#)
- [Encrypt site data](#)
- [Scale app](#)
 - [Scale up server capacity](#)
 - [Configure PremiumV3 tier](#)
 - [Scale out to multiple instances](#)
 - [High density hosting](#)
- [Monitor app](#)
 - [Monitor App Service with Azure Monitor](#)
 - [Quotas & alerts](#)
 - [Enable logs](#)
 - [Get resource events](#)
 - [Health check](#)
 - [Open SSH session to Linux container](#)
- [Manage app](#)
 - [Manage the hosting plan](#)
 - [Back up an app](#)
 - [Restore a backup](#)
 - [Restore a snapshot](#)
 - [Clone an app](#)
 - [Restore deleted App Service](#)
 - [Restore from downed region](#)
 - [Enable on Azure Arc](#)
 - [Enable VNet integration](#)
 - [Configure VNet integration routing](#)
- [Move app](#)
 - [Move between regions](#)

- [Move subscriptions](#)
- [Run background tasks](#)
 - [Create WebJobs](#)
 - [Develop WebJobs using VS](#)
 - [Get started with WebJobs SDK](#)
 - [Use WebJobs SDK](#)
- [High availability](#)
 - [Configuring high availability](#)
- [Automate provisioning](#)
 - [Bicep](#)
 - [Terraform](#)
- [Reference](#)
 - [Monitoring data](#)
 - [Azure CLI](#)
 - [Azure PowerShell](#)
 - [REST API](#)
 - [Resource Manager template](#)
 - [App settings reference](#)
 - [Azure Policy built-ins](#)
- [Resources](#)
 - [App Service Blog](#)
 - [Build your skills with Microsoft Learn](#)
 - [Azure Roadmap](#)
 - [Pricing](#)
 - [Quota Information](#)
 - [Service Updates](#)
 - [Kudu service](#)
 - [Migrate ASP.NET](#)
 - [Migrate Java](#)
 - [Best practices](#)
 - [Samples](#)
 - [Videos](#)

Cookbooks

[Reference Architectures](#)

[Deployment Scripts](#)

Troubleshooting

[Troubleshoot intermittent outbound connection errors](#)

[Troubleshoot with Visual Studio](#)

[Troubleshoot Node.js app](#)

[Troubleshoot HTTP 502 & 503](#)

[Troubleshoot performance issues](#)

[Troubleshoot domain and certificate issues](#)

FAQ

[Availability, performance, and application FAQ](#)

[Deployment FAQ](#)

[Open-source technologies FAQ](#)

[Configuration and management FAQ](#)

[App Service on Linux](#)

IP address change

[Inbound IP address](#)

[Outbound IP address](#)

[TLS/SSL address](#)

App Service overview

11/2/2021 • 5 minutes to read • [Edit Online](#)

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.

App Service not only adds the power of Microsoft Azure to your application, such as security, load balancing, autoscaling, and automated management. You can also take advantage of its DevOps capabilities, such as continuous deployment from Azure DevOps, GitHub, Docker Hub, and other sources, package management, staging environments, custom domain, and TLS/SSL certificates.

With App Service, you pay for the Azure compute resources you use. The compute resources you use are determined by the *App Service plan* that you run your apps on. For more information, see [Azure App Service plans overview](#).

Why use App Service?

Here are some key features of App Service:

- **Multiple languages and frameworks** - App Service has first-class support for ASP.NET, ASP.NET Core, Java, Ruby, Node.js, PHP, or Python. You can also run [PowerShell and other scripts or executables](#) as background services.
- **Managed production environment** - App Service automatically [patches and maintains the OS and language frameworks](#) for you. Spend time writing great apps and let Azure worry about the platform.
- **Containerization and Docker** - Dockerize your app and host a custom Windows or Linux container in App Service. Run multi-container apps with Docker Compose. Migrate your Docker skills directly to App Service.
- **DevOps optimization** - Set up [continuous integration and deployment](#) with Azure DevOps, GitHub, BitBucket, Docker Hub, or Azure Container Registry. Promote updates through [test and staging environments](#). Manage your apps in App Service by using [Azure PowerShell](#) or the [cross-platform command-line interface \(CLI\)](#).
- **Global scale with high availability** - Scale [up](#) or [out](#) manually or automatically. Host your apps anywhere in Microsoft's global datacenter infrastructure, and the App Service [SLA](#) promises high availability.
- **Connections to SaaS platforms and on-premises data** - Choose from more than 50 [connectors](#) for enterprise systems (such as SAP), SaaS services (such as Salesforce), and internet services (such as Facebook). Access on-premises data using [Hybrid Connections](#) and [Azure Virtual Networks](#).
- **Security and compliance** - App Service is [ISO](#), [SOC](#), and [PCI](#) compliant. Authenticate users with [Azure Active Directory](#), [Google](#), [Facebook](#), [Twitter](#), or [Microsoft account](#). Create [IP address restrictions](#) and [manage service identities](#).
- **Application templates** - Choose from an extensive list of application templates in the [Azure Marketplace](#), such as WordPress, Joomla, and Drupal.
- **Visual Studio and Visual Studio Code integration** - Dedicated tools in Visual Studio and Visual Studio Code streamline the work of creating, deploying, and debugging.
- **API and mobile features** - App Service provides turn-key CORS support for RESTful API scenarios, and simplifies mobile app scenarios by enabling authentication, offline data sync, push notifications, and more.
- **Serverless code** - Run a code snippet or script on-demand without having to explicitly provision or manage infrastructure, and pay only for the compute time your code actually uses (see [Azure Functions](#)).

Besides App Service, Azure offers other services that can be used for hosting websites and web applications. For

most scenarios, App Service is the best choice. For microservice architecture, consider [Azure Spring-Cloud Service](#) or [Service Fabric](#). If you need more control over the VMs on which your code runs, consider [Azure Virtual Machines](#). For more information about how to choose between these Azure services, see [Azure App Service, Virtual Machines, Service Fabric, and Cloud Services comparison](#).

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).

Built-in languages and frameworks

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. Run

`az webapp list-runtimes --linux` to view the latest languages and supported versions. If the runtime your application requires is not supported in the built-in images, you can deploy it with a custom container.

Outdated runtimes are periodically removed from the Web Apps Create and Configuration blades in the Portal. These runtimes are hidden from the Portal when they are deprecated by the maintaining organization or found to have significant vulnerabilities. These options are hidden to guide customers to the latest runtimes where they will be the most successful.

When an outdated runtime is hidden from the Portal, any of your existing sites using that version will continue to run. If a runtime is fully removed from the App Service platform, your Azure subscription owner(s) will receive an email notice before the removal.

If you need to create another web app with an outdated runtime version that is no longer shown on the Portal see the language configuration guides for instructions on how to get the runtime version of your site. You can use the Azure CLI to create another site with the same runtime. Alternatively, you can use the **Export Template** button on the web app blade in the Portal to export an ARM template of the site. You can reuse this template to deploy a new site with the same runtime and configuration.

Limitations

NOTE

Linux and Windows App Service plans can now share resource groups. This limitation has been lifted from the platform and existing resource groups have been updated to support this.

- App Service on Linux is not 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 may benefit from the custom container option, which places files in the container filesystem instead of on the content volume.

Next steps

Create your first web app.

[ASP.NET Core \(on Windows or Linux\)](#)

[ASP.NET \(on Windows\)](#)

[PHP \(on Windows or Linux\)](#)

[Ruby \(on Linux\)](#)

[Nodejs \(on Windows or Linux\)](#)

[Java \(on Windows or Linux\)](#)

[Python \(on Linux\)](#)

[HTML \(on Windows or Linux\)](#)

[Custom container \(Windows or Linux\)](#)

App Service Environment overview

11/2/2021 • 6 minutes to read • [Edit Online](#)

NOTE

This article is about the App Service Environment v3 which is used with Isolated v2 App Service plans

The Azure App Service Environment is an Azure App Service feature that provides a fully isolated and dedicated environment for securely running App Service apps at high scale. This capability can host your:

- Windows web apps
- Linux web apps
- Docker containers (Windows and Linux)
- Functions

App Service environments (ASEs) are appropriate for application workloads that require:

- High scale.
- Isolation and secure network access.
- High memory utilization.
- High requests per second (RPS). You can make multiple ASEs in a single Azure region or across multiple Azure regions. This flexibility makes an ASE ideal for horizontally scaling stateless applications with a high RPS requirement.

ASE's host applications from only one customer and do so in one of their VNets. Customers have fine-grained control over inbound and outbound application network traffic. Applications can establish high-speed secure connections over VPNs to on-premises corporate resources.

Usage scenarios

The App Service Environment has many use cases including:

- Internal line-of-business applications
- Applications that need more than 30 ASP instances
- Single tenant system to satisfy internal compliance or security requirements
- Network isolated application hosting
- Multi-tier applications

There are many networking features that enable apps in the multi-tenant App Service to reach network isolated resources or become network isolated themselves. These features are enabled at the application level. With an ASE, there's no added configuration required for the apps to be in the VNet. The apps are deployed into a network isolated environment that is already in a VNet. On top of the ASE hosting network isolated apps, it's also a single-tenant system. There are no other customers using the ASE. If you really need a complete isolation story, you can also get your ASE deployed onto dedicated hardware.

Dedicated environment

The ASE is a single tenant deployment of the Azure App Service that runs in your virtual network.

Applications are hosted in App Service plans, which are created in an App Service Environment. The App Service

plan is essentially a provisioning profile for an application host. As you scale your App Service plan out, you create more application hosts with all of the apps in that App Service plan on each host. A single ASEv3 can have up to 200 total App Service plan instances across all of the App Service plans combined. A single Isolated v2 App Service plan can have up to 100 instances by itself.

Virtual network support

The ASE feature is a deployment of the Azure App Service into a single subnet in a customer's Azure Resource Manager virtual network (VNet). When you deploy an app into an ASE, the app will be exposed on the inbound address assigned to the ASE. If your ASE is deployed with an internal VIP, then the inbound address for all of the apps will be an address in the ASE subnet. If your ASE is deployed with an external VIP, then the inbound address will be an internet addressable address and your apps will be in public DNS.

The number of addresses used by an ASEv3 in its subnet will vary based on how many instances you have along with how much traffic. There are infrastructure roles that are automatically scaled depending on the number of App Service plans and the load. The recommended size for your ASEv3 subnet is a /24 CIDR block with 256 addresses in it as that can host an ASEv3 scaled out to its limit.

The apps in an ASE do not need any features enabled to access resources in the same VNet that the ASE is in. If the ASE VNet is connected to another network, then the apps in the ASE can access resources in those extended networks. Traffic can be blocked by user configuration on the network.

The multi-tenant version of Azure App Service contains numerous features to enable your apps to connect to your various networks. Those networking features enable your apps to act as if they were deployed in a VNet. The apps in an ASEv3 do not need any configuration to be in the VNet. A benefit of using an ASE over the multi-tenant service is that any network access controls to the ASE hosted apps is external to the application configuration. With the apps in the multi-tenant service, you must enable the features on an app by app basis and use RBAC or policy to prevent any configuration changes.

Feature differences

Compared to earlier versions of the ASE, there are some differences with ASEv3. With ASEv3:

- There are no networking dependencies in the customer VNet. You can secure all inbound and outbound as desired. Outbound traffic can be routed also as desired.
- You can deploy it enabled for zone redundancy. Zone redundancy can only be set during ASEv3 creation and only in regions where all ASEv3 dependencies are zone redundant.
- You can deploy it on a dedicated host group. Host group deployments are not zone redundant.
- Scaling is much faster than with ASEv2. While scaling still is not immediate as in the multi-tenant service, it is a lot faster.
- Front end scaling adjustments are no longer required. The ASEv3 front ends automatically scale to meet needs and are deployed on better hosts.
- Scaling no longer blocks other scale operations within the ASEv3 instance. Only one scale operation can be in effect for a combination of OS and size. For example, while your Windows small App Service plan was scaling, you could kick off a scale operation to run at the same time on a Windows medium or anything else other than Windows small.
- Apps in an internal VIP ASEv3 can be reached across global peering. Access across global peering was not possible with ASEv2.

There are a few features that are not available in ASEv3 that were available in earlier versions of the ASE. In ASEv3, you can't:

- send SMTP traffic. You can still have email triggered alerts but your app can't send outbound traffic on port 25

- deploy your apps with FTP
- use remote debug with your apps
- upgrade yet from ASEv2
- monitor your traffic with Network Watcher or NSG Flow
- configure a IP-based TLS/SSL binding with your apps

Pricing

With ASEv3, there is a different pricing model depending on the type of ASE deployment you have. The three pricing models are:

- **ASEv3:** If ASE is empty, there is a charge as if you had one ASP with one instance of Windows I1v2. The one instance charge is not an additive charge but is only applied if the ASE is empty.
- **Availability Zone ASEv3:** There is a minimum nine Windows I1v2 instance charge. There is no added charge for availability zone support if you have nine or more App Service plan instances. All App Service plans in an AZ ASEv3 also have a minimum instance count of 3 to ensure there is an instance in each availability zone. As the plans are scaled out, they are spread across the availability zones.
- **Dedicated host ASEv3:** With a dedicated host deployment, you are charged for two dedicated hosts per our pricing at ASEv3 creation then a small percentage of the Isolated V2 rate per core charge as you scale.

Reserved Instance pricing for Isolated v2 is available and is described in [How reservation discounts apply to Azure App Service](#). The pricing, along with reserved instance pricing, is available at [App Service pricing](#) under **Isolated v2 plan**.

Regions

The ASEv3 is available in the following regions.

NORMAL AND DEDICATED HOST ASEV3 REGIONS	AZ ASEV3 REGIONS
Australia East	Australia East
Australia Southeast	Brazil South
Brazil South	Canada Central
Canada Central	Central US
Central India	East US
Central US	East US 2
East Asia	France Central
East US	Germany West Central
East US 2	Japan East
France Central	North Europe
Germany West Central	South Central US

NORMAL AND DEDICATED HOST ASEV3 REGIONS	AZ ASEV3 REGIONS
Japan East	Southeast Asia
Korea Central	UK South
North Central US	West Europe
North Europe	West US 2
Norway East	
South Africa North	
South Central US	
Southeast Asia	
Switzerland North	
UAE North	
UK South	
UK West	
West Central US	
West Europe	
West US	
West US 2	
West US 3	

Quickstart: Deploy an ASP.NET web app

11/2/2021 • 14 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to create and deploy your first ASP.NET web app to [Azure App Service](#). App Service supports various versions of .NET apps, and provides a highly scalable, self-patching web hosting service. ASP.NET web apps are cross-platform and can be hosted on Linux or Windows. When you're finished, you'll have an Azure resource group consisting of an App Service hosting plan and an App Service with a deployed web application.

NOTE

Azure PowerShell is recommended for creating apps on the Windows hosting platform. To create apps on Linux, use a different tool, such as [Azure CLI](#)

Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- [Visual Studio 2019](#) with the **ASP.NET and web development** workload.

If you've already installed Visual Studio 2019:

- Install the latest updates in Visual Studio by selecting **Help > Check for Updates**.
 - Add the workload by selecting **Tools > Get Tools and Features**.
- An Azure account with an active subscription. [Create an account for free](#).
 - [Visual Studio Code](#).
 - The [Azure Tools](#) extension.
- [.NET 5.0](#)
 - [.NET Framework 4.8](#)

Install the latest .NET 5.0 SDK.

- An Azure account with an active subscription. [Create an account for free](#).
 - The [Azure CLI](#).
 - The .NET SDK (includes runtime and CLI).
- [.NET 5.0](#)
 - [.NET Framework 4.8](#)

Install the latest .NET 5.0 SDK.

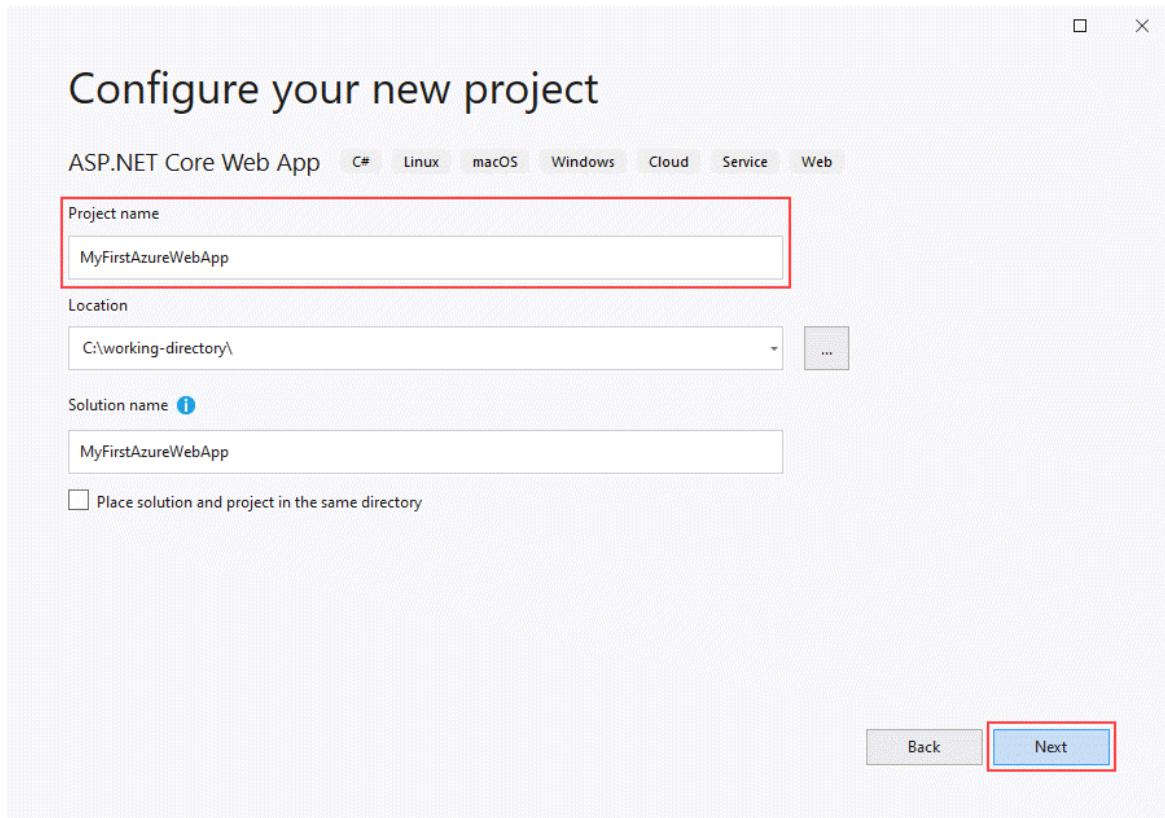
- An Azure account with an active subscription. [Create an account for free](#).
 - The [Azure PowerShell](#).
 - The .NET SDK (includes runtime and CLI).
- [.NET 5.0](#)
 - [.NET Framework 4.8](#)

Install the latest .NET 5.0 SDK.

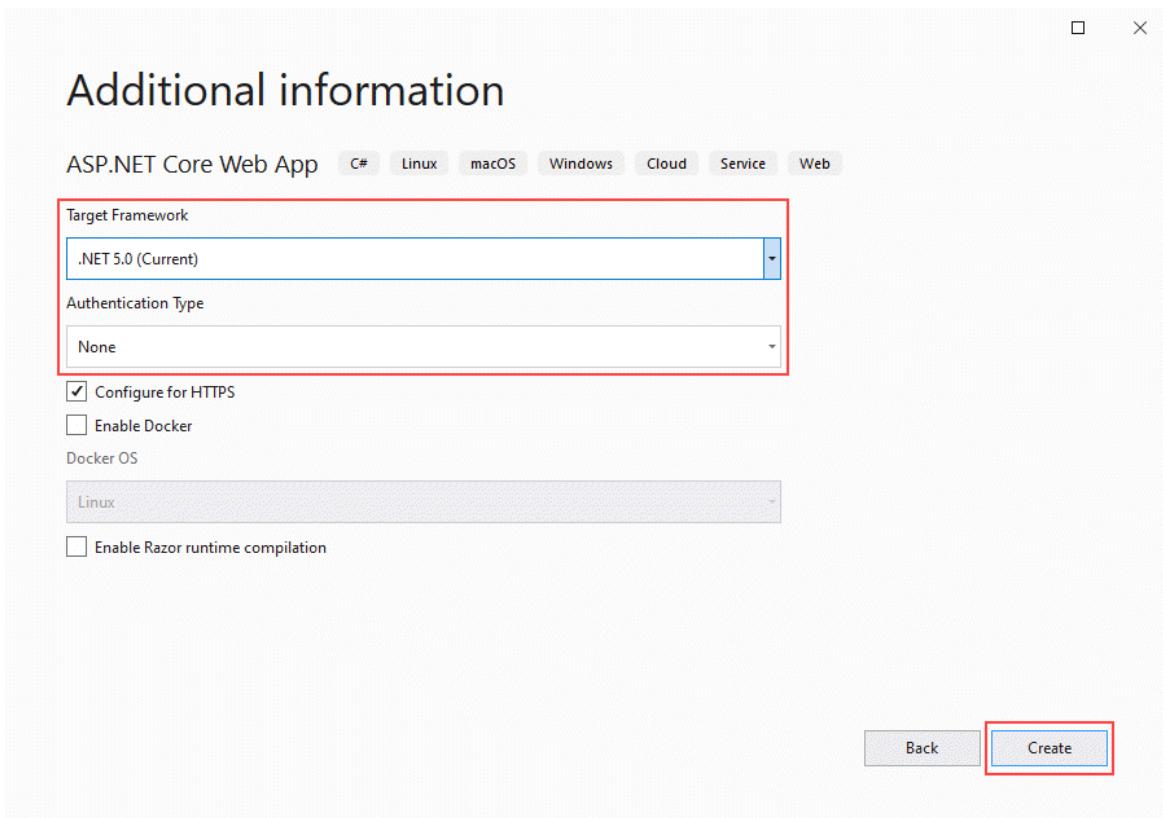
Create an ASP.NET web app

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

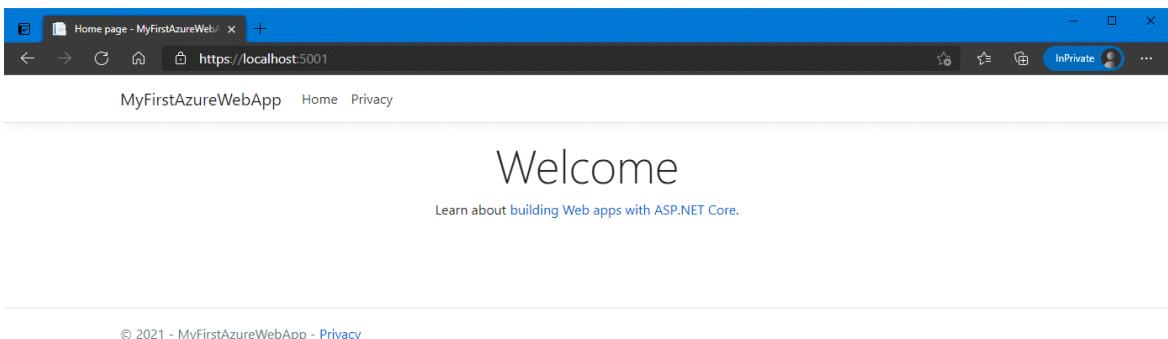
1. Open Visual Studio and then select **Create a new project**.
2. In **Create a new project**, find, and choose **ASP.NET Core Web App**, then select **Next**.
3. In **Configure your new project**, name the application *MyFirstAzureWebApp*, and then select **Next**.



4. Select **.NET Core 5.0 (Current)**.
5. Make sure **Authentication Type** is set to **None**. Select **Create**.



- From the Visual Studio menu, select **Debug > Start Without Debugging** to run the web app locally.



Create a new folder named *MyFirstAzureWebApp*, and open it in Visual Studio Code. Open the **Terminal** window, and create a new .NET web app using the `dotnet new webapp` command.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

```
dotnet new webapp -f net5.0
```

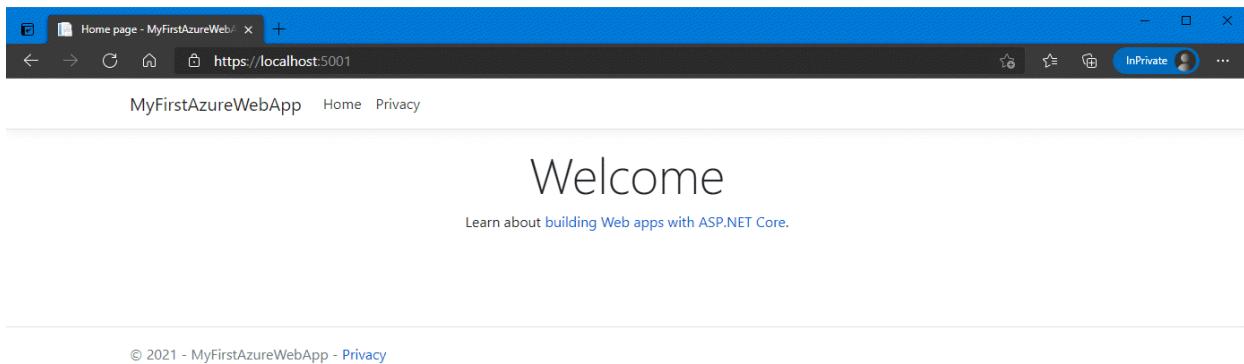
From the **Terminal** in Visual Studio Code, run the application locally using the `dotnet run` command.

```
dotnet run
```

Open a web browser, and navigate to the app at `https://localhost:5001`.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

You'll see the template ASP.NET Core 5.0 web app displayed in the page.



Open a terminal window on your machine to a working directory. Create a new .NET web app using the `dotnet new webapp` command, and then change directories into the newly created app.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

```
dotnet new webapp -n MyFirstAzureWebApp --framework net5.0
cd MyFirstAzureWebApp
```

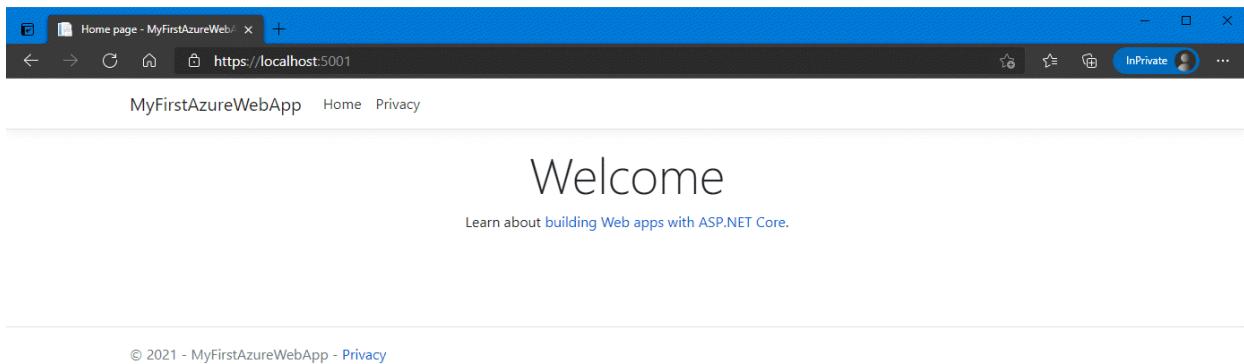
From the same terminal session, run the application locally using the `dotnet run` command.

```
dotnet run
```

Open a web browser, and navigate to the app at <https://localhost:5001>.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

You'll see the template ASP.NET Core 5.0 web app displayed in the page.



Publish your web app

To publish your web app, you must first create and configure a new App Service that you can publish your app to.

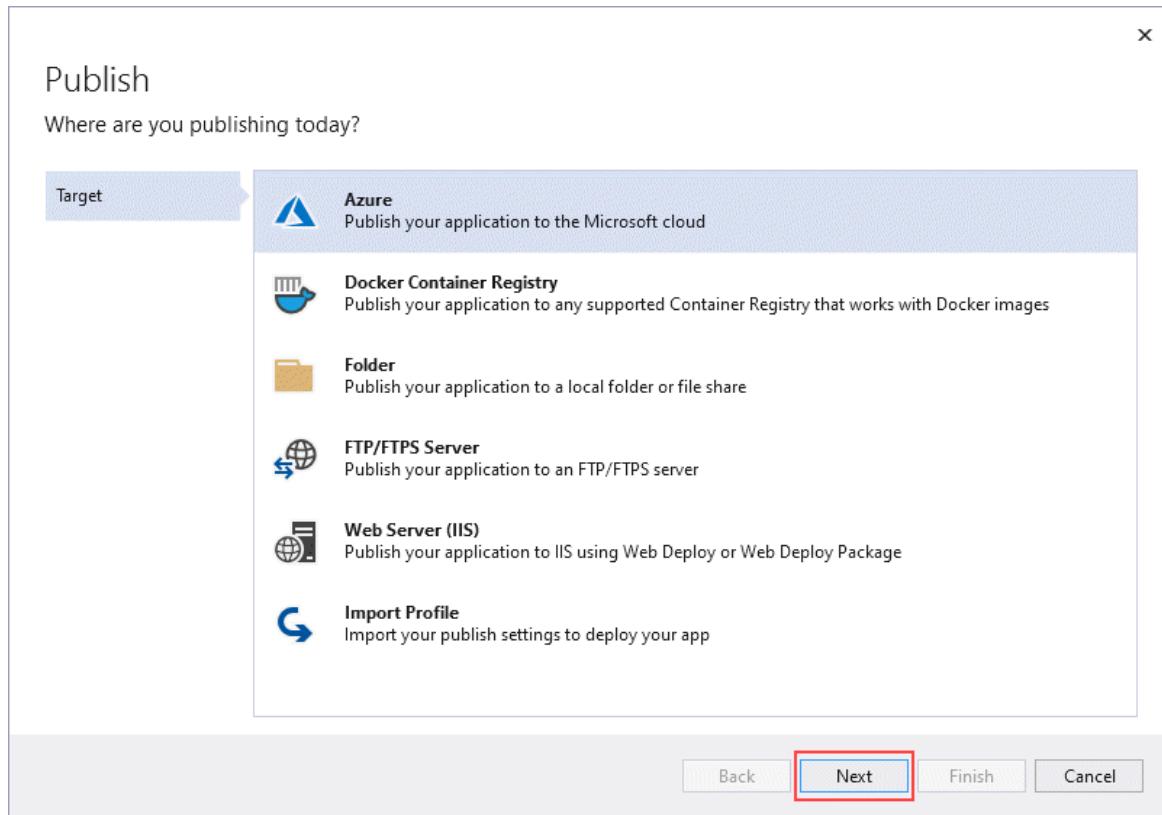
As part of setting up the App Service, you'll create:

- A new [resource group](#) to contain all of the Azure resources for the service.
- A new [Hosting Plan](#) that specifies the location, size, and features of the web server farm that hosts your app.

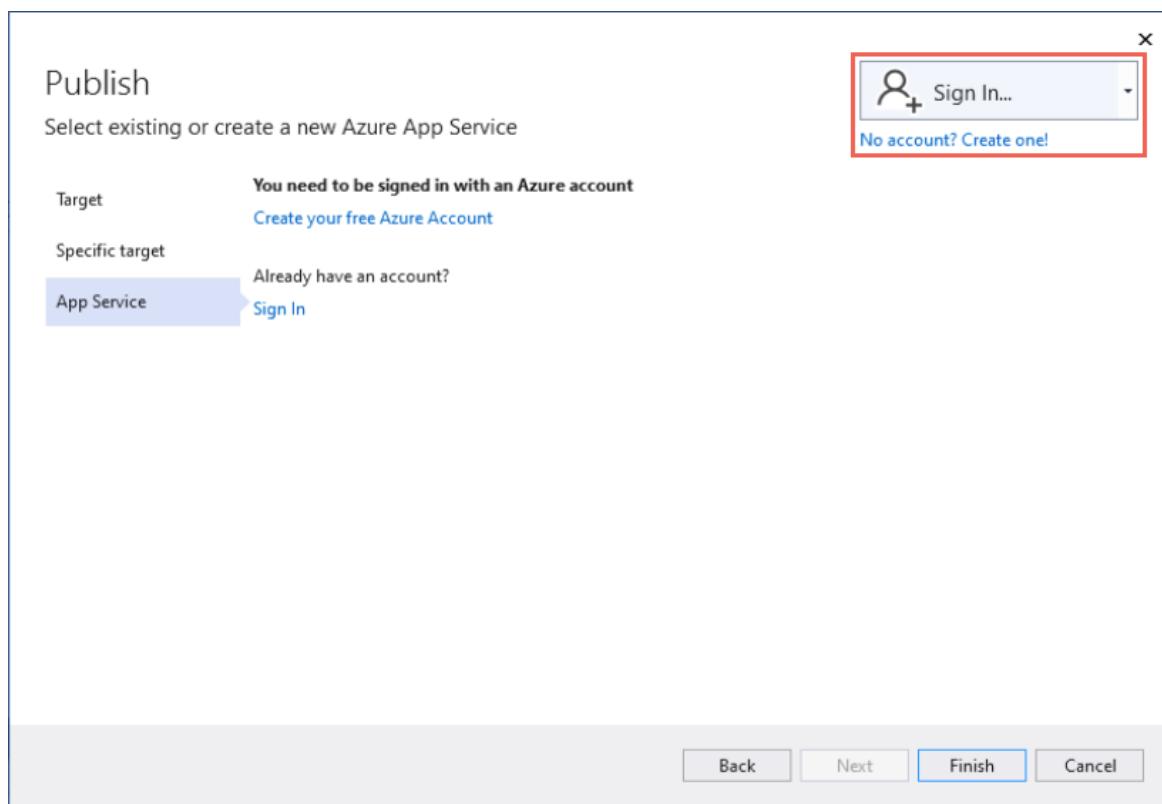
Follow these steps to create your App Service and publish your web app:

1. In [Solution Explorer](#), right-click the **MyFirstAzureWebApp** project and select **Publish**.

2. In Publish, select Azure and then Next.



3. Your options depend on whether you're signed in to Azure already and whether you have a Visual Studio account linked to an Azure account. Select either Add an account or Sign in to sign in to your Azure subscription. If you're already signed in, select the account you want.

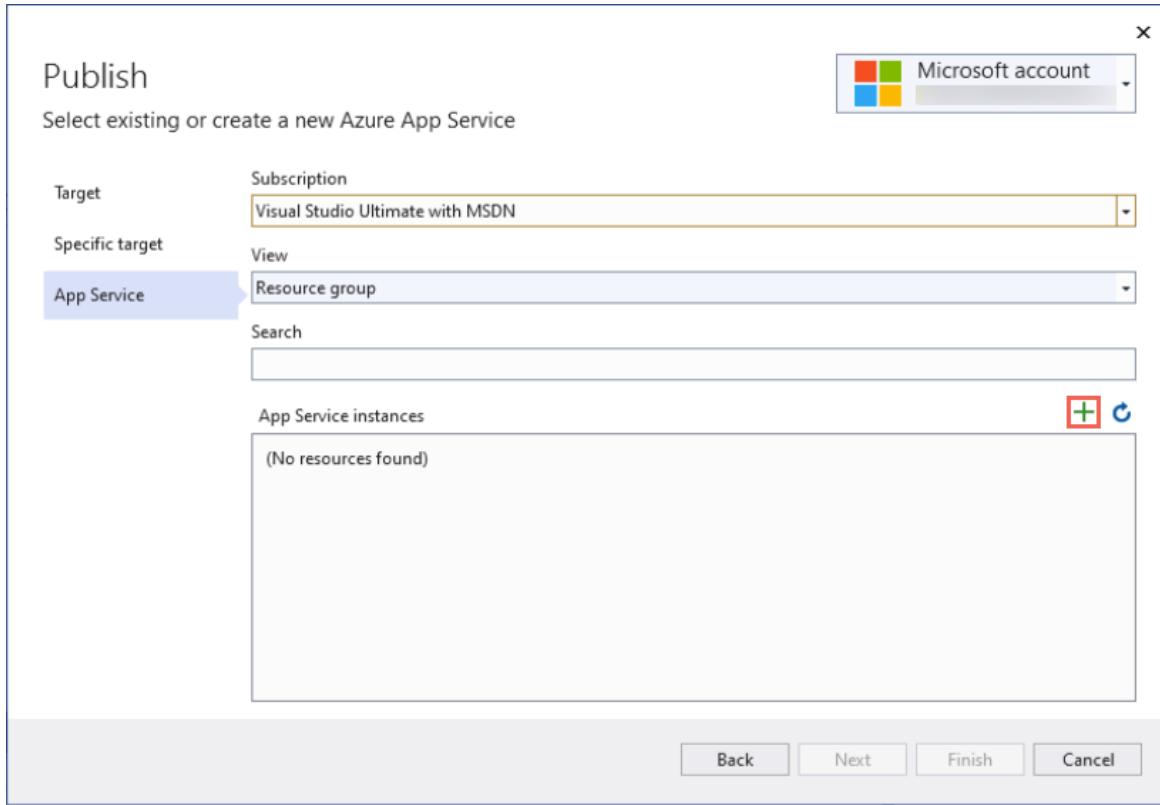


4. Choose the Specific target, either Azure App Service (Linux) or Azure App Service (Windows).

IMPORTANT

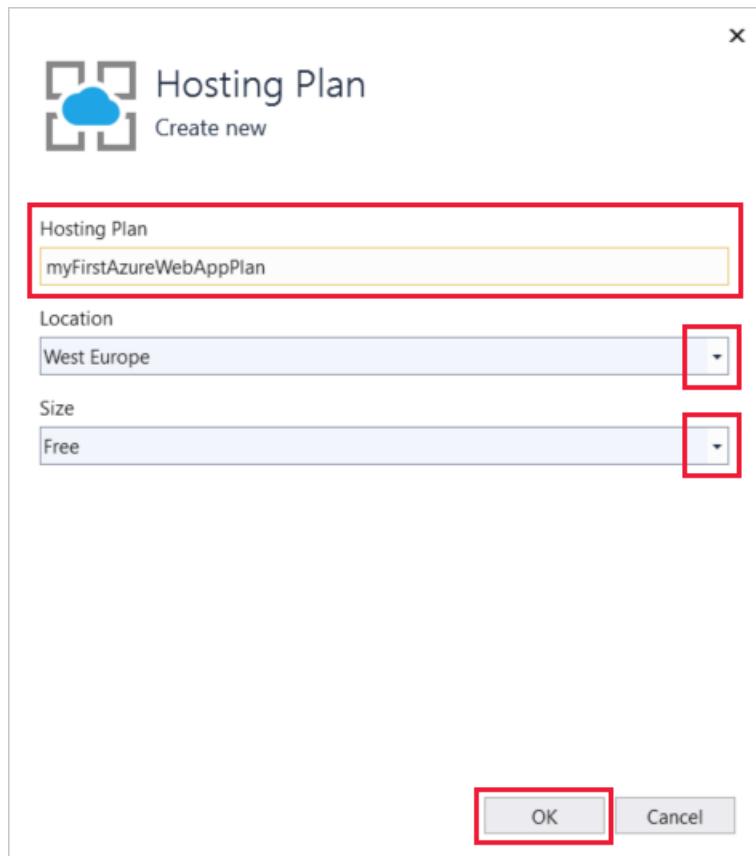
When targeting ASP.NET Framework 4.8, you will use **Azure App Service (Windows)**.

5. To the right of **App Service instances**, select +.



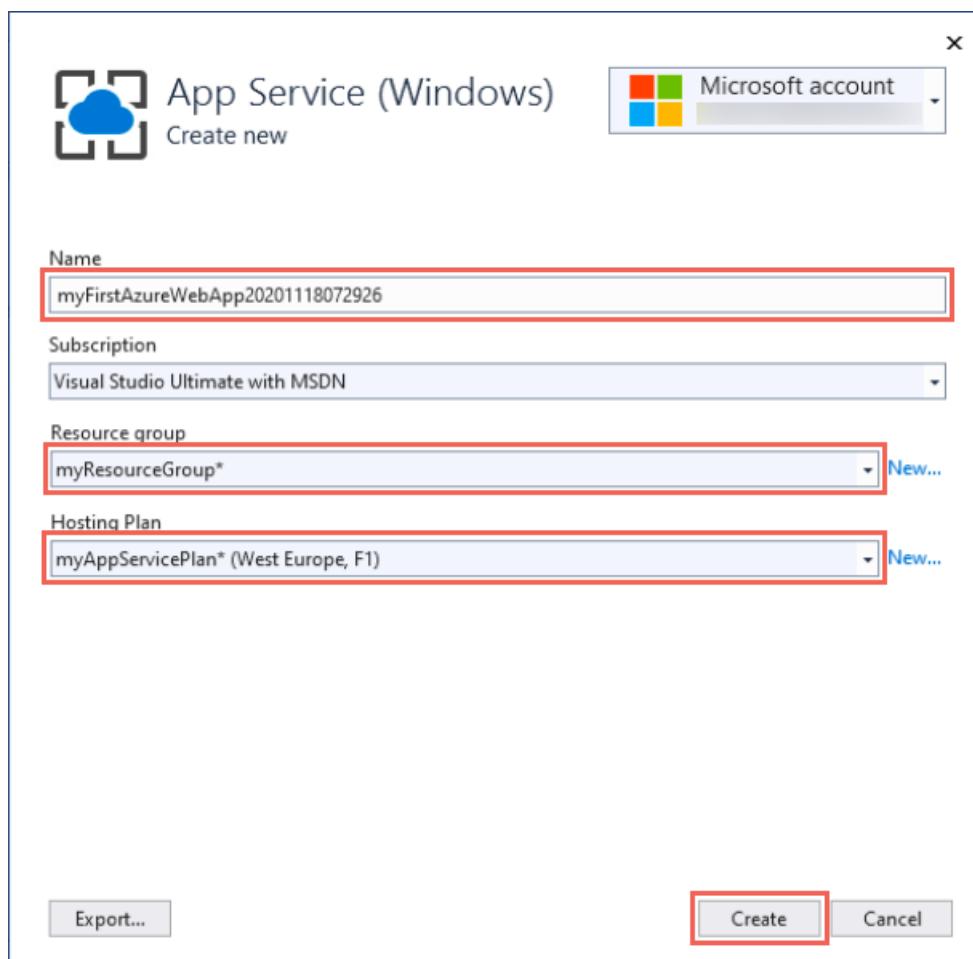
6. For **Subscription**, accept the subscription that is listed or select a new one from the drop-down list.
7. For **Resource group**, select **New**. In **New resource group name**, enter *myResourceGroup* and select **OK**.
8. For **Hosting Plan**, select **New**.
9. In the **Hosting Plan: Create new** dialog, enter the values specified in the following table:

SETTING	SUGGESTED VALUE	DESCRIPTION
Hosting Plan	<i>MyFirstAzureWebAppPlan</i>	Name of the App Service plan.
Location	<i>West Europe</i>	The datacenter where the web app is hosted.
Size	<i>Free</i>	Pricing tier determines hosting features.



10. In **Name**, enter a unique app name that includes only the valid characters are `a-z`, `A-Z`, `0-9`, and `-`. You can accept the automatically generated unique name. The URL of the web app is `http://<app-name>.azurewebsites.net`, where `<app-name>` is your app name.

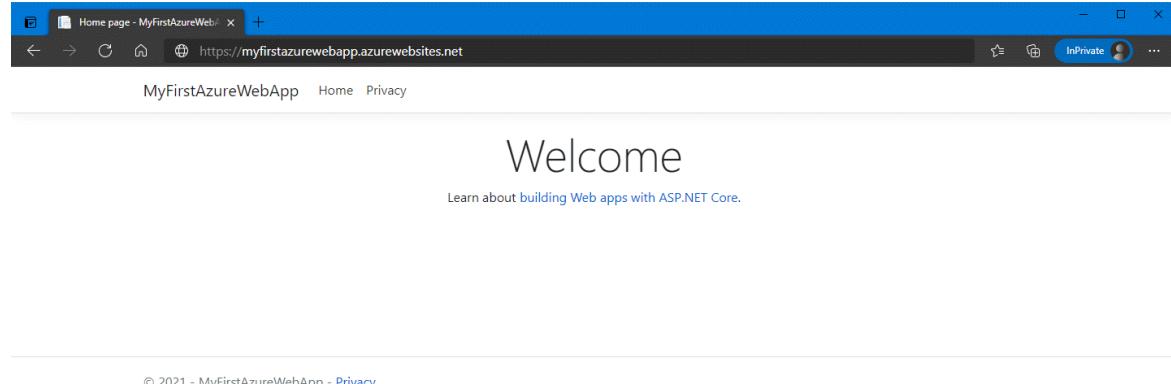
11. Select **Create** to create the Azure resources.



Once the wizard completes, the Azure resources are created for you and you are ready to publish.

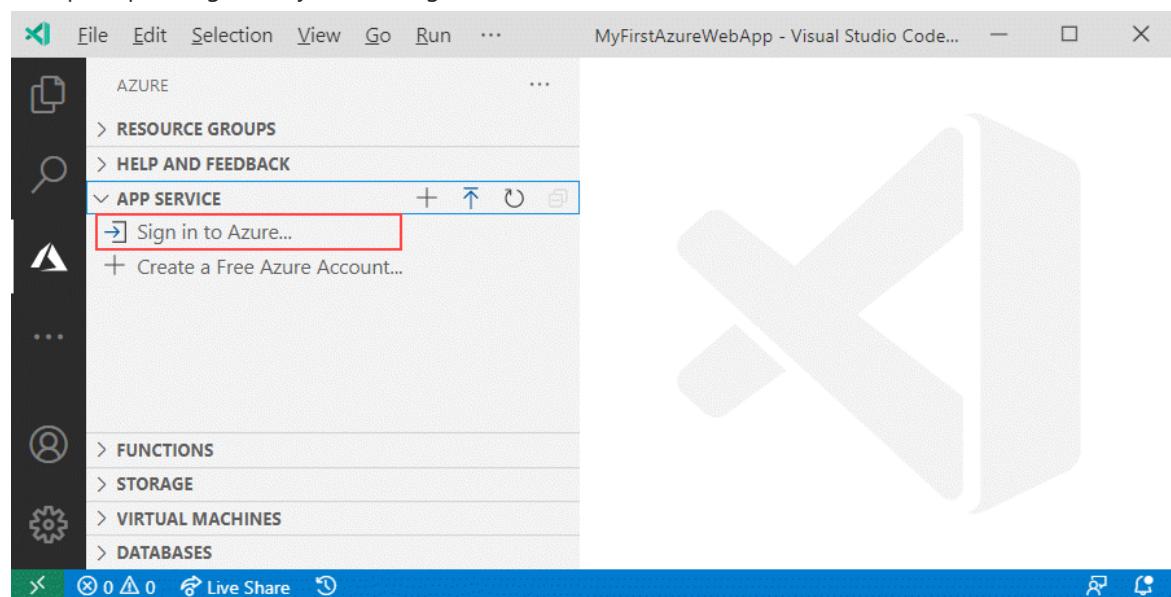
12. Select **Finish** to close the wizard.
13. In the **Publish** page, select **Publish**. Visual Studio builds, packages, and publishes the app to Azure, and then launches the app in the default browser.
 - [.NET 5.0](#)
 - [.NET Framework 4.8](#)

You'll see the ASP.NET Core 5.0 web app displayed in the page.



To deploy your web app using the Visual Studio Azure Tools extension:

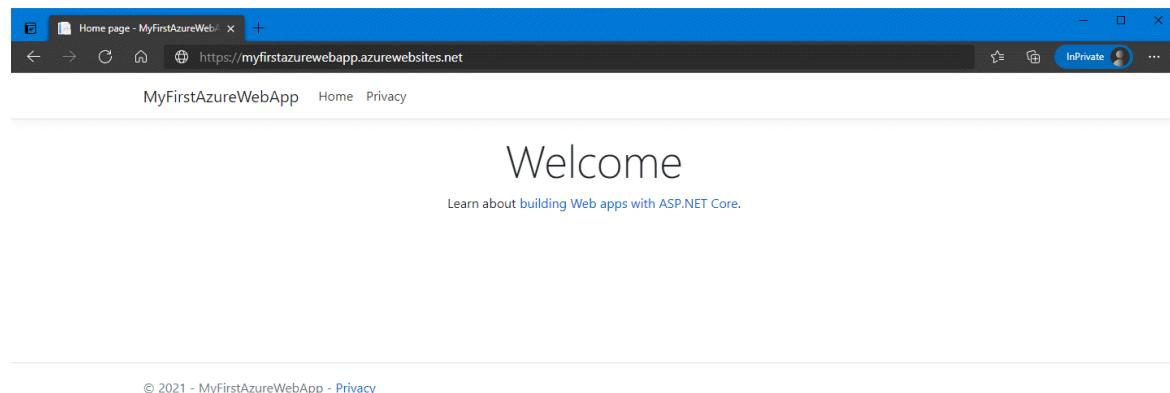
1. In Visual Studio Code, open the **Command Palette**, **Ctrl+Shift+P**.
2. Search for and select "Azure App Service: Deploy to Web App".
3. Respond to the prompts as follows:
 - Select *MyFirstAzureWebApp* as the folder to deploy.
 - Select **Add Config** when prompted.
 - If prompted, sign in to your existing Azure account.



- Select your **Subscription**.
- Select **Create new Web App... Advanced**.
- For **Enter a globally unique name**, use a name that's unique across all of Azure (*valid characters are a-z , 0-9 , and -*). A good pattern is to use a combination of your company name and an app identifier.

- Select **Create new resource group** and provide a name like `myResourceGroup`.
 - When prompted to **Select a runtime stack**:
 - For *.NET 5.0*, select **.NET 5**
 - For *.NET Framework 4.8*, select **ASP.NET V4.8**
 - Select an operating system (Windows or Linux).
 - For *.NET Framework 4.8*, Windows will be selected implicitly.
 - Select **Create a new App Service plan**, provide a name, and select the **F1 Free pricing tier**.
 - Select **Skip for now** for the Application Insights resource.
 - Select a location near you.
4. When publishing completes, select **Browse Website** in the notification and select **Open** when prompted.
- **.NET 5.0**
 - **.NET Framework 4.8**

You'll see the ASP.NET Core 5.0 web app displayed in the page.



1. Sign into your Azure account by using the `az login` command and following the prompt:

```
az login
```

2. Deploy the code in your local *MyFirstAzureWebApp* directory using the `az webapp up` command:

```
az webapp up --sku F1 --name <app-name> --os-type <os>
```

- If the `az` command isn't recognized, be sure you have the Azure CLI installed as described in [Prerequisites](#).
- Replace `<app-name>` with a name that's unique across all of Azure (*valid characters are a-z, 0-9, and -*). A good pattern is to use a combination of your company name and an app identifier.
- The `--sku F1` argument creates the web app on the **Free pricing tier**. Omit this argument to use a faster premium tier, which incurs an hourly cost.
- Replace `<os>` with either `linux` or `windows`. You must use `windows` when targeting *ASP.NET Framework 4.8*.
- You can optionally include the argument `--location <location-name>` where `<location-name>` is an available Azure region. You can retrieve a list of allowable regions for your Azure account by running the `az account list-locations` command.

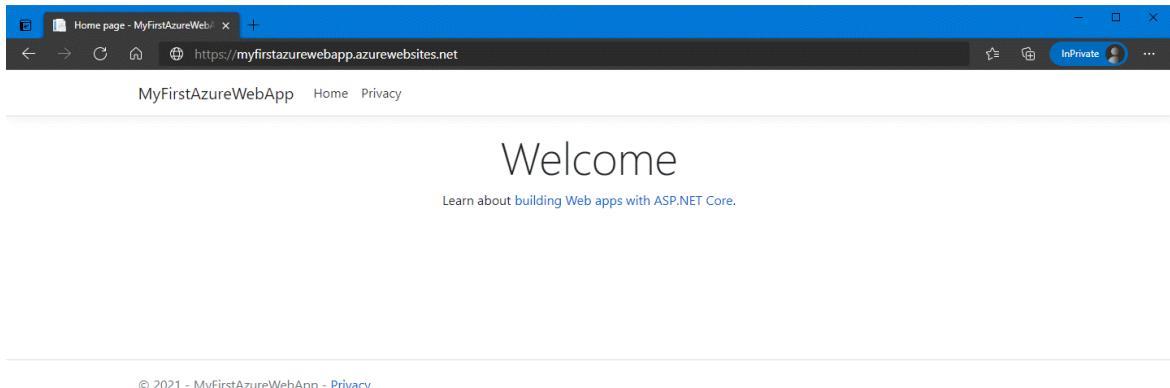
The command may take a few minutes to complete. While running, it provides messages about creating the resource group, the App Service plan, and hosting app, configuring logging, then performing ZIP deployment. It then outputs a message with the app's URL:

You can launch the app at <http://<app-name>.azurewebsites.net>

3. Open a web browser and navigate to the URL:

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

You'll see the ASP.NET Core 5.0 web app displayed in the page.



NOTE

Azure PowerShell is recommended for creating apps on the Windows hosting platform. To create apps on Linux, use a different tool, such as [Azure CLI](#)

1. Sign into your Azure account by using the `Connect-AzAccount` command and following the prompt:

```
Connect-AzAccount
```

2. Create a new app by using the `New-AzWebApp` command:

```
New-AzWebApp -Name <app-name> -Location westeurope
```

- Replace `<app-name>` with a name that's unique across all of Azure (*valid characters are a-z, 0-9, and -*). A good pattern is to use a combination of your company name and an app identifier.
- You can optionally include the parameter `-Location <location-name>` where `<location-name>` is an available Azure region. You can retrieve a list of allowable regions for your Azure account by running the `Get-AzLocation` command.

The command may take a few minutes to complete. While running, it creates a resource group, an App Service plan, and the App Service resource.

3. From the application root folder, prepare your local *MyFirstAzureWebApp* application for deployment using the `dotnet publish` command:

```
dotnet publish --configuration Release
```

4. Change to the release directory and create a zip file from the contents:

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

```
cd bin\Release\net5.0\publish  
Compress-Archive -Path * -DestinationPath deploy.zip
```

5. Publish the zip file to the Azure app using the [Publish-AzWebApp](#) command:

```
Publish-AzWebApp -ResourceGroupName myResourceGroup -Name <app-name> -ArchivePath (Get-Item .\deploy.zip).FullName -Force
```

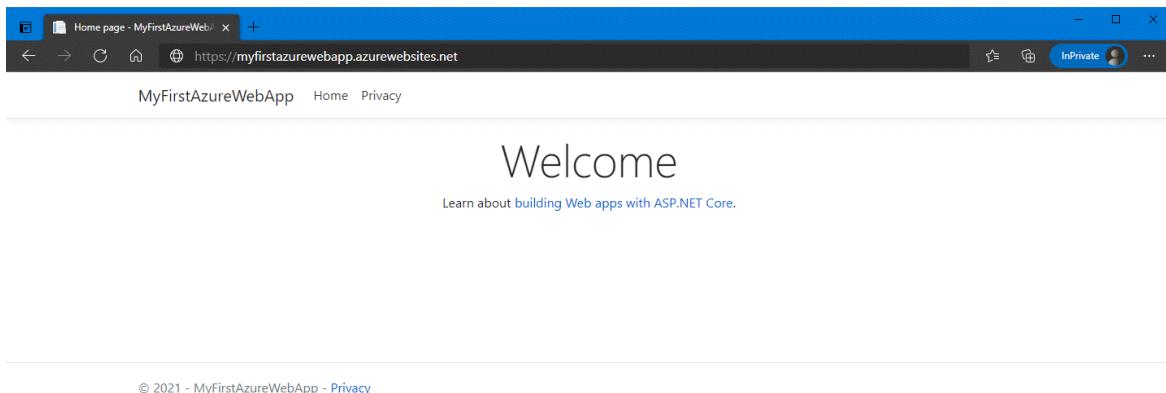
NOTE

`-ArchivePath` needs the full path of the zip file.

6. Open a web browser and navigate to the URL:

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

You'll see the ASP.NET Core 5.0 web app displayed in the page.



Update the app and redeploy

Follow these steps to update and redeploy your web app:

1. In **Solution Explorer**, under your project, open *Index.cshtml*.
2. Replace the first `<div>` element with the following code:

```
<div class="jumbotron">  
    <h1>.NET 5 Azure</h1>  
    <p class="lead">Example .NET app to Azure App Service.</p>  
</div>
```

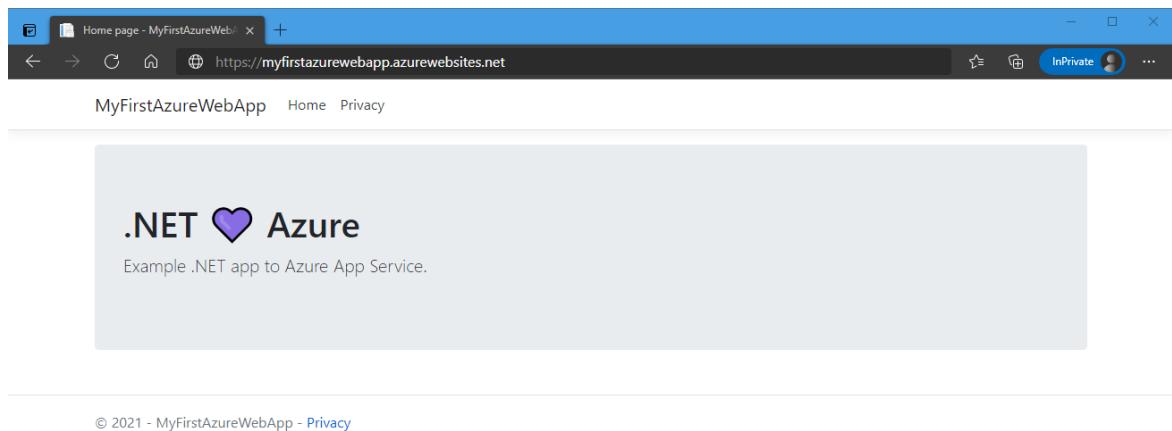
Save your changes.

3. To redeploy to Azure, right-click the **MyFirstAzureWebApp** project in **Solution Explorer** and select **Publish**.
4. In the **Publish** summary page, select **Publish**.

When publishing completes, Visual Studio launches a browser to the URL of the web app.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

You'll see the updated ASP.NET Core 5.0 web app displayed in the page.



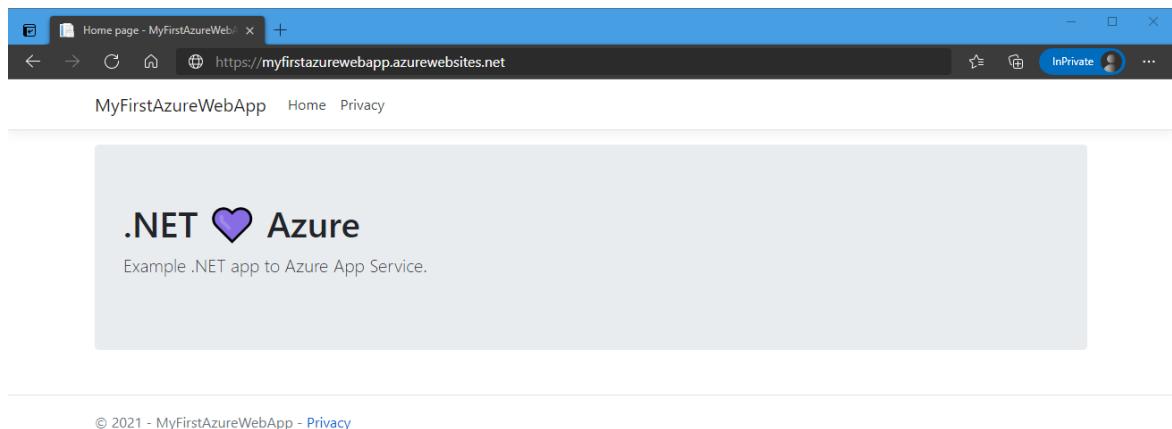
1. Open *Index.cshtml*.
2. Replace the first `<div>` element with the following code:

```
<div class="jumbotron">
    <h1>.NET ❤️ Azure</h1>
    <p class="lead">Example .NET app to Azure App Service.</p>
</div>
```

Save your changes.

3. Open the Visual Studio Code **Side Bar**, select the **Azure** icon to expand its options.
4. Under the **APP SERVICE** node, expand your subscription and right-click on the **MyFirstAzureWebApp**.
5. Select the **Deploy to Web App....**
6. Select **Deploy** when prompted.
7. When publishing completes, select **Browse Website** in the notification and select **Open** when prompted.
 - [.NET 5.0](#)
 - [.NET Framework 4.8](#)

You'll see the updated ASP.NET Core 5.0 web app displayed in the page.



In the local directory, open the *Index.cshtml* file. Replace the first `<div>` element:

```
<div class="jumbotron">
    <h1>.NET ❤ Azure</h1>
    <p class="lead">Example .NET app to Azure App Service.</p>
</div>
```

Save your changes, then redeploy the app using the `az webapp up` command again:

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

ASP.NET Core 5.0 is cross-platform, based on your previous deployment replace `<os>` with either `linux` or `windows`.

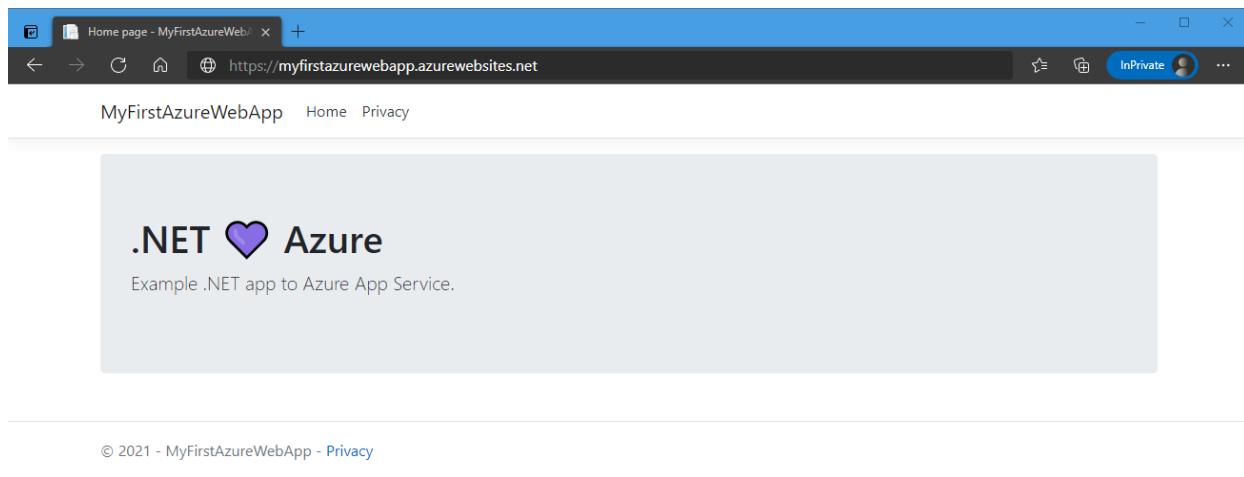
```
az webapp up --os-type <os>
```

This command uses values that are cached locally in the `.azure/config` file, including the app name, resource group, and App Service plan.

Once deployment has completed, switch back to the browser window that opened in the **Browse to the app** step, and hit refresh.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

You'll see the updated ASP.NET Core 5.0 web app displayed in the page.



1. In the local directory, open the `/Index.cshtml` file. Replace the first `<div>` element:

```
<div class="jumbotron">
    <h1>.NET ❤ Azure</h1>
    <p class="lead">Example .NET app to Azure App Service.</p>
</div>
```

2. From the application root folder, prepare your local `MyFirstAzureWebApp` application for deployment using the `dotnet publish` command:

```
dotnet publish --configuration Release
```

3. Change to the release directory and create a zip file from the contents:

- [.NET 5.0](#)

- .NET Framework 4.8

```
cd bin\Release\net5.0\publish
Compress-Archive -Path * -DestinationPath deploy.zip
```

4. Publish the zip file to the Azure app using the [Publish-AzWebApp](#) command:

```
Publish-AzWebApp -ResourceGroupName myResourceGroup -Name <app-name> -ArchivePath (Get-Item .\deploy.zip).FullName -Force
```

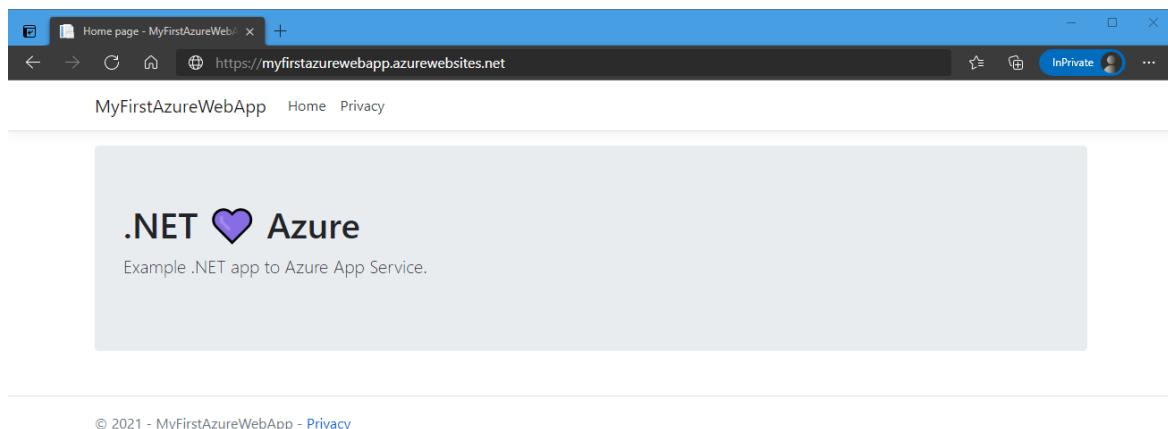
NOTE

-ArchivePath needs the full path of the zip file.

5. Once deployment has completed, switch back to the browser window that opened in the [Browse to the app](#) step, and hit refresh.

- .NET 5.0
- .NET Framework 4.8

You'll see the updated ASP.NET Core 5.0 web app displayed in the page.



Manage the Azure app

To manage your web app, go to the [Azure portal](#), and search for and select [App Services](#).

On the App Services page, select the name of your web app.

The screenshot shows the Azure portal's App Services blade. At the top, there are filter options for 'Subscription == all', 'Resource group == all', and 'Location == all'. Below the filters, it says 'Showing 1 to 2 of 2 records.' There are five columns: 'Name' (with a sorting arrow), 'Status' (sorted by 'Running'), 'Location' (sorted by 'Central US'), 'Pricing Tier' (sorted by 'Free'), and 'App Service Plan' (sorted by 'username_asp_Linux_centralus_0'). The first row shows 'my-demo-app' with 'Running' status, 'Central US' location, 'Free' pricing tier, and 'username_asp_Linux_centralus_0' app service plan. The second row shows 'myFirstAzureWebApp20201123082420' with 'Running' status, 'West Europe' location, 'Free' pricing tier, and 'myAppServicePlan' app service plan. A red box highlights the second row.

The **Overview** page for your web app, contains options for basic management like browse, stop, start, restart, and delete. The left menu provides further pages for configuring your app.

The screenshot shows the Azure portal's Overview page for the web app 'myFirstAzureWebApp20201123082420'. The left sidebar has a navigation menu with links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Events (preview), Deployment (selected), Quickstart, Deployment slots, Deployment Center, and Deployment Center (Preview). The main pane shows the app's details: Resource group (myResourceGroup), Status (Running), Location (West Europe), Subscription (Visual Studio Ultimate with MSDN), Subscription ID (00000000-0000-0000-0000-000000000000), and Tags (Click here to add tags). It also includes sections for 'Diagnose and solve problems' and 'Application Insights'.

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group.

1. From your web app's **Overview** page in the Azure portal, select the **myResourceGroup** link under **Resource group**.
2. On the resource group page, make sure that the listed resources are the ones you want to delete.
3. Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group.

1. From your web app's **Overview** page in the Azure portal, select the **myResourceGroup** link under **Resource group**.
2. On the resource group page, make sure that the listed resources are the ones you want to delete.
3. Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following PowerShell command:

```
Remove-AzResourceGroup -Name myResourceGroup
```

This command may take a minute to run.

Next steps

In this quickstart, you created and deployed an ASP.NET web app to Azure App Service.

- [.NET 5.0](#)
- [.NET Framework 4.8](#)

Advance to the next article to learn how to create a .NET Core app and connect it to a SQL Database:

[Tutorial: ASP.NET Core app with SQL database](#)

[Configure ASP.NET Core app](#)

Create a Node.js web app in Azure

11/2/2021 • 12 minutes to read • [Edit Online](#)

In this quickstart, you'll learn how to create and deploy your first Node.js ([Express](#)) web app to [Azure App Service](#). App Service supports various versions of Node.js on both Linux and Windows.

This quickstart configures an App Service app in the **Free** tier and incurs no cost for your Azure subscription.

Set up your initial environment

- Have an Azure account with an active subscription. [Create an account for free](#).
- Install [Node.js and npm](#). Run the command `node --version` to verify that Node.js is installed.
- Install [Visual Studio Code](#).
- The [Azure App Service extension](#) for Visual Studio Code.
- Have an Azure account with an active subscription. [Create an account for free](#).
- Install [Node.js and npm](#). Run the command `node --version` to verify that Node.js is installed.
- Install [Azure CLI](#), with which you run commands in any shell to provision and configure Azure resources.
- Have an Azure account with an active subscription. [Create an account for free](#).
- Install [Node.js and npm](#). Run the command `node --version` to verify that Node.js is installed.
- Have a FTP client (for example, [FileZilla](#)), to connect to your app.

Create your Node.js application

In this step, you create a starter Node.js application and make sure it runs on your computer.

TIP

If you have already completed the [Node.js tutorial](#), you can skip ahead to [Deploy to Azure](#).

1. Create a simple Node.js application using the [Express Generator](#), which is installed by default with Node.js and NPM.

```
npx express-generator myExpressApp --view pug
```

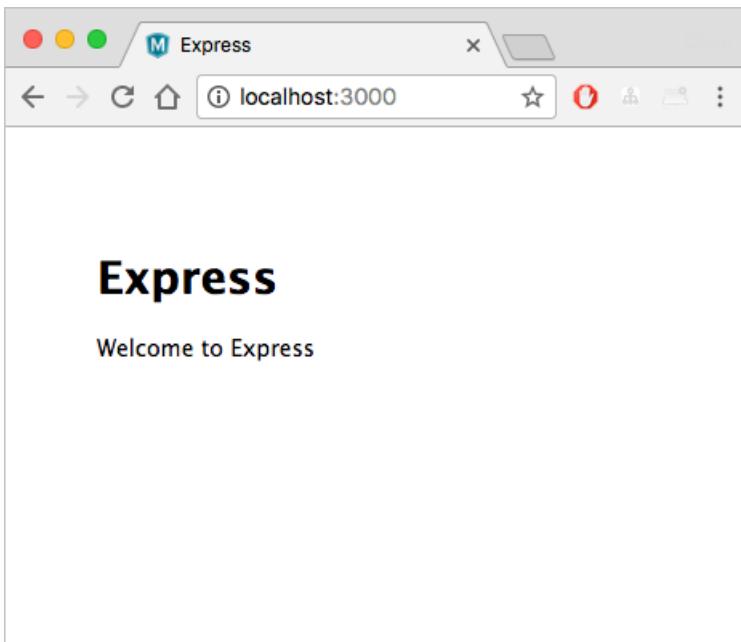
2. Change to the application's directory and install the NPM packages.

```
cd myExpressApp  
npm install
```

3. Start the development server.

```
npm start
```

4. In a browser, navigate to `http://localhost:3000`. You should see something like this:



I ran into an issue

Deploy to Azure

Before you continue, ensure that you have all the prerequisites installed and configured.

NOTE

For your Node.js application to run in Azure, it needs to listen on the port provided by the `PORT` environment variable. In your generated Express app, this environment variable is already used in the startup script `bin/www` (search for `process.env.PORT`).

Sign in to Azure

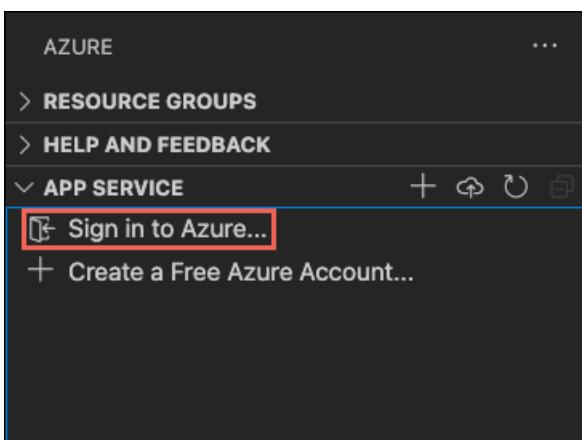
1. In the terminal, make sure you're in the `myExpressApp` directory, then start Visual Studio Code with the following command:

```
code .
```

2. In Visual Studio Code, in the [Activity Bar](#), select the **Azure** logo.

3. In the **App Service** explorer, select **Sign in to Azure...** and follow the instructions.

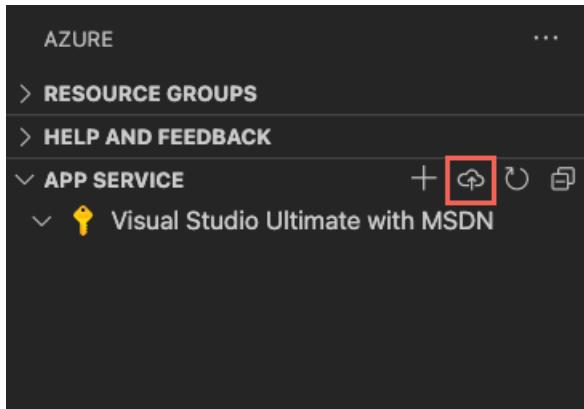
In Visual Studio Code, you should see your Azure email address in the Status Bar and your subscription in the **AZURE APP SERVICE** explorer.



I ran into an issue

Configure the App Service app and deploy code

1. In the App Service explorer, select the Deploy to Web App icon.



2. Choose the *myExpressApp* folder.

- [Deploy to Linux](#)
- [Deploy to Windows](#)

3. Choose **Create new Web App**. A Linux container is used by default.

4. Type a globally unique name for your web app and press **Enter**. The name must be unique across all of Azure and use only alphanumeric characters ('A-Z', 'a-z', and '0-9') and hyphens ('-').
5. In Select a runtime stack, select the Node.js version you want. An **LTS** version is recommended.
6. In Select a pricing tier, select **Free (F1)** and wait for the the resources to be provisioned in Azure.
7. In the popup **Always deploy the workspace "myExpressApp" to <app-name>**, select **Yes**. This way, as long as you're in the same workspace, Visual Studio Code deploys to the same App Service app each time.

While Visual Studio Code provisions the Azure resources and deploys the code, it shows [progress notifications](#).

8. Once deployment completes, select **Browse Website** in the notification popup. The browser should display the Express default page.

I ran into an issue

In the terminal, make sure you're in the *myExpressApp* directory, and deploy the code in your local folder (*myExpressApp*) using the `az webapp up` command:

- [Deploy to Linux](#)
- [Deploy to Windows](#)

```
az webapp up --sku F1 --name <app-name>
```

- If the `az` command isn't recognized, be sure you have the Azure CLI installed as described in [Set up your initial environment](#).
- Replace `<app_name>` with a name that's unique across all of Azure (*valid characters are a-z, 0-9, and -*). A good pattern is to use a combination of your company name and an app identifier.
- The `--sku F1` argument creates the web app on the Free pricing tier, which incurs a no cost.

- You can optionally include the argument `--location <location-name>` where `<location_name>` is an available Azure region. You can retrieve a list of allowable regions for your Azure account by running the `az account list-locations` command.
- The command creates a Linux app for Node.js by default. To create a Windows app instead, use the `--os-type` argument.
- If you see the error, "Could not auto-detect the runtime stack of your app," make sure you're running the command in the `myExpressApp` directory (See [Troubleshooting auto-detect issues with az webapp up](#)).

The command may take a few minutes to complete. While running, it provides messages about creating the resource group, the App Service plan, and the app resource, configuring logging, and doing ZIP deployment. It then gives the message, "You can launch the app at `http://<app-name>.azurewebsites.net`", which is the app's URL on Azure.

```
The webapp '<app-name>' doesn't exist
Creating Resource group '<group-name>' ...
Resource group creation complete
Creating AppServicePlan '<app-service-plan-name>' ...
Creating webapp '<app-name>' ...
Configuring default logging for the app, if not already enabled
Creating zip with contents of dir /home/cephas/myExpressApp ...
Getting scm site credentials for zip deployment
Starting zip deployment. This operation can take a while to complete ...
Deployment endpoint responded with status code 202
You can launch the app at http://<app-name>.azurewebsites.net
{
  "URL": "http://<app-name>.azurewebsites.net",
  "appserviceplan": "<app-service-plan-name>",
  "location": "centralus",
  "name": "<app-name>",
  "os": "<os-type>",
  "resourcegroup": "<group-name>",
  "runtime_version": "node|10.14",
  "runtime_version_detected": "0.0",
  "sku": "FREE",
  "src_path": "//home//cephas//myExpressApp"
}
```

NOTE

The `az webapp up` command does the following actions:

- Create a default [resource group](#).
- Create a default [App Service plan](#).
- [Create an app](#) with the specified name.
- [Zip deploy](#) all files from the current working directory, [with build automation enabled](#).
- Cache the parameters locally in the `.azure/config` file so that you don't need to specify them again when deploying later with `az webapp up` or other Azure CLI commands. The cached values are used automatically by default.

Sign in to Azure portal

Sign in to the Azure portal at <https://portal.azure.com>.

Create Azure resources

1. Type **app services** in the search. Under **Services**, select **App Services**.

2. In the App Services page, select Create.
3. In the Basics tab, under Project details, make sure the correct subscription is selected and then choose to Create new resource group. Type *myResourceGroup* for the name.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	<input type="text" value="Pay-As-You-Go"/>
Resource Group *	<input type="text" value="(New) myResourceGroup"/> Create new

4. Under Instance details, type a globally unique name for your web app and select Code. Choose Node 14 LTS Runtime stack, an Operating System, and a Region you want to serve your app from.

Instance Details

Need a database? [Try the new Web + Database experience.](#)

Name *	<input type="text" value="myNodeApp"/> .azurewebsites.net
Publish *	<input checked="" type="radio"/> Code <input type="radio"/> Docker Container
Runtime stack *	<input type="text" value="Node 14 LTS"/>
Operating System *	<input checked="" type="radio"/> Linux <input type="radio"/> Windows
Region *	<input type="text" value="Central US"/> <small>Not finding your App Service Plan? Try a different region.</small>

5. Under App Service Plan, choose to Create new App Service Plan. Type *myAppServicePlan* for the name.

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#)

Linux Plan (Central US) *	<input type="text" value="(New) myAppServicePlan"/> Create new
Sku and size *	Free F1 1 GB memory Change size

6. Select the Review + create button at the bottom of the page.

[Review + create](#)

< Previous

Next : Deployment >

7. After validation runs, select the Create button at the bottom of the page.

8. After deployment is complete, select Go to resource.

^ Next steps

[Manage deployments for your app.](#) Recommended

[Protect your app with authentication.](#) Recommended

[Go to resource](#)

Get FTP credentials

Azure App Service supports **two types of credentials** for FTP/S deployment. These credentials are not the same as your Azure subscription credentials. In this section, you get the *application-scope credentials* to use with FileZilla.

1. From the App Service app page, click **Deployment Center** in the left-hand menu and select **FTPS credentials** tab.

The screenshot shows the Azure App Service Deployment Center for the 'myNodeApp' app service. The 'FTPS credentials' tab is selected. The 'FTPS endpoint' field contains the URL 'https://waws-prod-drm1-207.ftp.azurewebsites.windows.net/site/wwwroot'. The 'Application scope' section notes that auto-generated credentials provide access only to the specific app or deployment slot. The 'Username' field is populated with 'myNodeApp\myNodeApp' and the 'Password' field is filled with a long string of characters. The 'Deployment' section on the left has 'Deployment Center' highlighted with a red box.

2. Open **FileZilla** and create a new site.

3. From the **FTPS credentials** tab, copy **FTPS endpoint**, **Username**, and **Password** into FileZilla.

The screenshot shows the 'Transfer Settings' tab of the FileZilla configuration dialog. The 'Protocol' is set to 'FTP - File Transfer Protocol'. The 'Host' is 'azurewebsites.windows.net/site/wwwroot' and the 'Port' is blank. The 'Encryption' setting is 'Require explicit FTP over TLS'. The 'Logon Type' is 'Normal', 'User' is 'myNodeApp\myNodeApp', and the 'Password' is masked with dots. Other tabs like General, Advanced, and Charset are visible at the top.

4. Click **Connect** in FileZilla.

Deploy files with FTP

1. Copy all files and directories files to the [/site/wwwroot](#) directory in Azure.

Filename	Files...	Filetype	Last modified
..			
bin		File folder	11/1/2021 10:47:00 AM
node_modules		File folder	11/1/2021 10:50:00 AM
public		File folder	11/1/2021 10:58:00 AM
routes		File folder	11/1/2021 10:58:00 AM
views		File folder	11/1/2021 10:58:00 AM
app.js	1,074	JavaScript File	11/1/2021 10:47:00 AM
hostingstart.html	3,499	Microsoft Edge HTML Document	10/26/2021 2:16:00 PM
package-lock.json	75,249	JSON File	11/1/2021 10:47:00 AM
package.json	304	JSON File	11/1/2021 10:47:00 AM

2. Browse to your app's URL to verify the app is running properly.

Redeploy updates

You can deploy changes to this app by making edits in Visual Studio Code, saving your files, and then redeploy to your Azure app. For example:

1. From the sample project, open `views/index.pug` and change

```
p Welcome to #{title}
```

to

```
p Welcome to Azure!
```

2. In the App Service explorer, select the Deploy to Web App icon again, confirm by clicking Deploy again.
3. Wait for deployment to complete, then select Browse Website in the notification popup. You should see that the `Welcome to Express` message has been changed to `Welcome to Azure!`.
2. Save your changes, then redeploy the app using the `az webapp up` command again with no arguments:

```
az webapp up
```

This command uses values that are cached locally in the `.azure/config` file, such as the app name, resource group, and App Service plan.

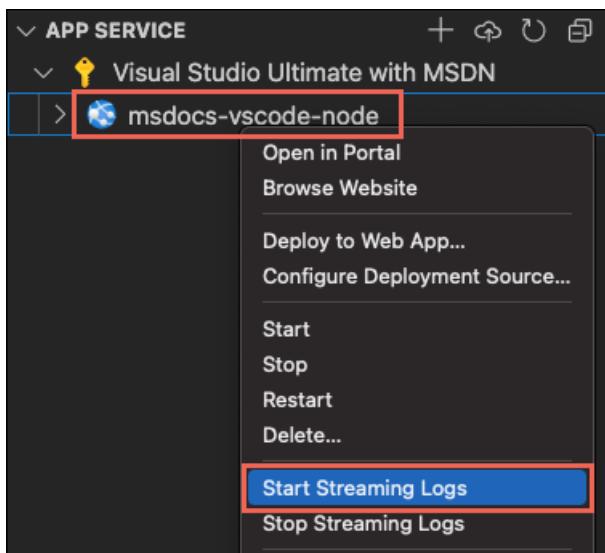
3. Once deployment is complete, refresh the webpage `http://<app-name>.azurewebsites.net`. You should see that the `Welcome to Express` message has been changed to `Welcome to Azure!`.
2. Save your changes, then redeploy the app using your FTP client again.
3. Once deployment is complete, refresh the webpage `http://<app-name>.azurewebsites.net`. You should see that the `Welcome to Express` message has been changed to `Welcome to Azure!`.

Stream Logs

You can stream log output (calls to `console.log()`) from the Azure app directly in the Visual Studio Code output

window.

1. In the App Service explorer, right-click the app node and choose Start Streaming Logs.



2. If asked to restart the app, click Yes. Once the app is restarted, the Visual Studio Code output window opens with a connection to the log stream.
3. After a few seconds, the output window shows a message indicating that you're connected to the log-streaming service. You can generate more output activity by refreshing the page in the browser.

```
Connecting to log stream...
2020-03-04T19:29:44 Welcome, you are now connected to log-streaming service. The default
timeout is 2 hours.
Change the timeout with the App Setting SCM_LOGSTREAM_TIMEOUT (in seconds).
```

I ran into an issue

You can access the console logs generated from inside the app and the container in which it runs. Logs include any output generated by calls to `console.log()`.

To stream logs, run the `az webapp log tail` command:

```
az webapp log tail
```

The command uses the resource group name cached in the `.azure/config` file.

You can also include the `--logs` parameter with the `az webapp up` command to automatically open the log stream on deployment.

Refresh the app in the browser to generate console logs, which include messages describing HTTP requests to the app. If no output appears immediately, try again in 30 seconds.

To stop log streaming at any time, press **Ctrl+C** in the terminal.

You can access the console logs generated from inside the app and the container in which it runs. You can stream log output (calls to `console.log()`) from the Node.js app directly in the Azure portal.

1. In the same App Service page for your app, use the left menu to scroll to the *Monitoring* section and click **Log stream**.

The screenshot shows the Azure Log Stream interface for the 'myNodeApp' application. On the left, there's a navigation sidebar with sections like Monitoring, Automation, and Support + troubleshooting. Under Monitoring, the 'Log stream' option is highlighted with a red box. The main area displays a terminal-like window titled 'Starting Live Log Stream ---'. The log output includes several messages related to the Node.js application's startup and configuration, such as documentation links, NodeJS quickstart, NodeJS Version, and environment variable notes. The logs end with a message about writing an output script to '/opt/startup/startup.sh'.

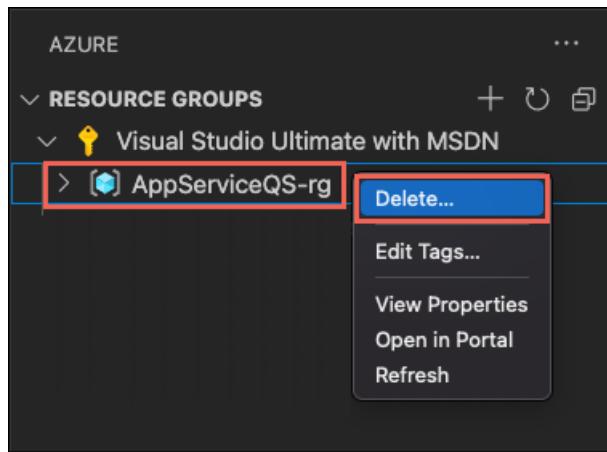
- After a few seconds, the output window shows a message indicating that you're connected to the log-streaming service. You can generate more output activity by refreshing the page in the browser.

```
Connecting...
2021-10-26T21:04:14 Welcome, you are now connected to log-streaming service.
Starting Log Tail -n 10 of existing logs ---
/appsvctmp/volatile/logs/runtime/81b1b83b27ea1c3d598a1cdec28c71c4074ce66c735d0be57f15a8d0
7cb3178e.log
2021-10-26T21:04:08.614384810Z: [INFO]
2021-10-26T21:04:08.614393710Z: [INFO] # Enter the source directory to make sure the
script runs where the user expects
2021-10-26T21:04:08.614399010Z: [INFO] cd "/home/site/wwwroot"
2021-10-26T21:04:08.614403210Z: [INFO]
2021-10-26T21:04:08.614407110Z: [INFO] export
NODE_PATH=/usr/local/lib/node_modules:$NODE_PATH
2021-10-26T21:04:08.614411210Z: [INFO] if [ -z "$PORT" ]; then
2021-10-26T21:04:08.614415310Z: [INFO]     export PORT=8080
2021-10-26T21:04:08.614419610Z: [INFO] fi
2021-10-26T21:04:08.614423411Z: [INFO]
2021-10-26T21:04:08.614427211Z: [INFO] node /opt/startup/default-static-site.js
Ending Log Tail of existing logs ---
```

Clean up resources

In the preceding steps, you created Azure resources in a resource group. The create steps in this quickstart put all the resources in this resource group. To clean up, you just need to remove the resource group.

- In the Azure extension of Visual Studio, expand the **Resource Groups** explorer.
- Expand the subscription, right-click the resource group you created earlier, and select **Delete**.



3. When prompted, confirm your deletion by entering the name of the resource group you're deleting. Once you confirm, the resource group is deleted, and you see a [notification](#) when it's done.

I ran into an issue

In the preceding steps, you created Azure resources in a resource group. The resource group has a name like "appsvc_rg_Linux_CentralUS" depending on your location.

If you don't expect to need these resources in the future, delete the resource group by running the following command:

```
az group delete --no-wait
```

The command uses the resource group name cached in the `.azure/config` file.

The `--no-wait` argument allows the command to return before the operation is complete.

When no longer needed, you can delete the resource group, App service, and all related resources.

1. From your App Service *overview* page, click the *resource group* you created in the [Create Azure resources](#) step.

Home >

myNodeApp

App Service

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Events (preview)

Browse Stop Swap Restart Delete

Resource group (change) : myResourceGroup

Status : Running

Location : Central US

Subscription (change) : {your subscription name}

Subscription ID : {your subscription id}

Tags (Edit) : Click here to add tags

2. From the *resource group* page, select **Delete resource group**. Confirm the name of the resource group to finish deleting the resources.

The screenshot shows the Azure portal interface for a resource group named 'myResourceGroup'. The top navigation bar includes 'Search (Ctrl+ /)', 'Create', 'Edit columns', and a redboxed 'Delete resource group' button. On the left, a sidebar lists 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Resource visualizer', and 'Events'. The main content area is titled 'Essentials' and displays subscription information: 'Subscription (Move) : {your subscription name}', 'Subscription ID : {your subscription id}', and 'Tags (Edit) : Click here to add tags'. Below this is a 'Resources' section with a link to 'Recommendations (4)'. A red box highlights the 'Delete resource group' button.

Next steps

Congratulations, you've successfully completed this quickstart!

[Tutorial: Node.js app with MongoDB](#)

[Configure Nodejs app](#)

Check out the other Azure extensions.

- [Cosmos DB](#)
- [Azure Functions](#)
- [Docker Tools](#)
- [Azure CLI Tools](#)
- [Azure Resource Manager Tools](#)

Or get them all by installing the [Node Pack for Azure](#) extension pack.

Create a PHP web app in Azure App Service

11/2/2021 • 8 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This quickstart tutorial shows how to deploy a PHP app to Azure App Service on Windows.

Azure App Service provides a highly scalable, self-patching web hosting service. This quickstart tutorial shows how to deploy a PHP app to Azure App Service on Linux.

You create the web app using the [Azure CLI](#) in Cloud Shell, and you use Git to deploy sample PHP code to the web app.



You can follow the steps here using a Mac, Windows, or Linux machine. Once the prerequisites are installed, it takes about five minutes to complete the steps.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

To complete this quickstart:

- [Install Git](#)
- [Install PHP](#)

Download the sample locally

1. In a terminal window, run the following commands. This will clone the sample application to your local machine, and navigate to the directory containing the sample code.

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

2. Make sure the default branch is `main`.

```
git branch -m main
```

TIP

The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main`, this quickstart also shows you how to deploy a repository from `main`.

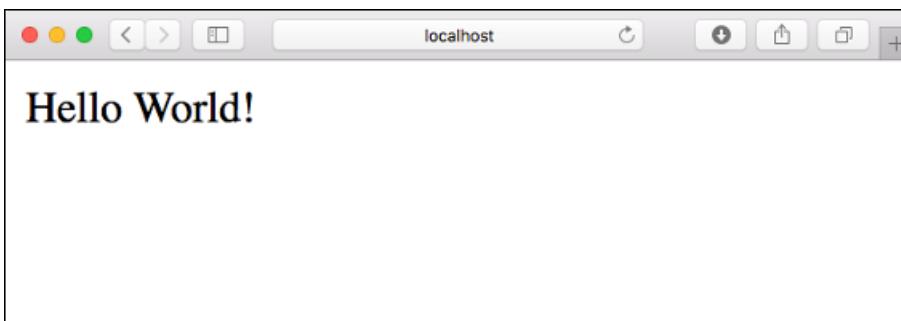
Run the app locally

1. Run the application locally so that you see how it should look when you deploy it to Azure. Open a terminal window and use the `php` command to launch the built-in PHP web server.

```
php -S localhost:8080
```

2. Open a web browser, and navigate to the sample app at <http://localhost:8080>.

You see the **Hello World!** message from the sample app displayed in the page.



3. In your terminal window, press **Ctrl+C** to exit the web server.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	A screenshot of a code block. At the top right, there is a "Try It" button with a red box drawn around it. Below the code block, there is a "Copy" button.
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	A screenshot of the Azure portal's top navigation bar. A blue button labeled "Launch Cloud Shell" is highlighted with a red box.
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	A screenshot of the Azure portal's top navigation bar. The "Cloud Shell" icon (represented by a square with a diagonal line) is highlighted with a red box.

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.

3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.

4. Select **Enter** to run the code.

Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the [az webapp deployment user set](#) command in Azure Cloud Shell. Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the [az group create](#) command. The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service in **Free** tier, run the [az appservice list-locations --sku FREE](#) command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the [az group create](#) command. The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service on Linux in **Basic** tier, run the [az appservice list-locations --sku B1 --linux-workers-enabled](#) command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

Create an Azure App Service plan

In the Cloud Shell, create an App Service plan with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
  "freeOfferExpirationTime": null,
  "geoRegion": "West Europe",
  "hostingEnvironmentProfile": null,
  "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
  "kind": "linux",
  "location": "West Europe",
  "maximumNumberOfWorkers": 1,
  "name": "myAppServicePlan",
  < JSON data removed for brevity. >
  "targetWorkerSizeId": 0,
  "type": "Microsoft.Web/serverfarms",
  "workerTierName": null
}
```

Create a web app

1. In the Cloud Shell, create a web app in the `myAppServicePlan` App Service plan with the `az webapp create` command.

In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `PHP|7.4`. To see all supported runtimes, run `az webapp list-runtimes`

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
'PHP|7.4' --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```

Local git is configured with url of 'https://<username>@<app-
name>.scm.azurewebsites.net/<app-name>.git'
{
  "availabilityState": "Normal",
  "clientAffinityEnabled": true,
  "clientCertEnabled": false,
  "cloningInfo": null,
  "containerSize": 0,
  "dailyMemoryTimeQuota": 0,
  "defaultHostName": "<app-name>.azurewebsites.net",
  "enabled": true,
  < JSON data removed for brevity. >
}

```

You've created an empty new web app, with git deployment enabled.

NOTE

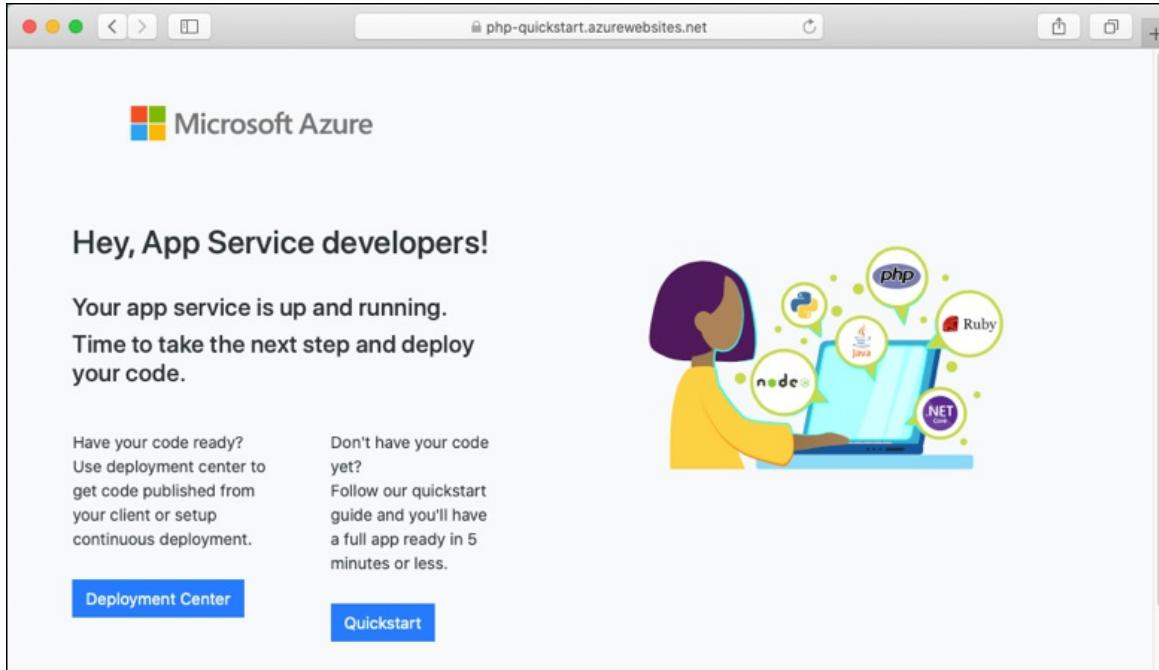
The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

`https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

2. Browse to your newly created web app. Replace `<app-name>` with your unique app name created in the prior step.

`http://<app-name>.azurewebsites.net`

Here is what your new web app should look like:



Push to Azure from Git

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings  
DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace *<deploymentLocalGitUrl-from-create-step>* with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in [Configure a deployment user](#), not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

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

```
Counting objects: 2, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (2/2), 352 bytes | 0 bytes/s, done.  
Total 2 (delta 1), reused 0 (delta 0)  
remote: Updating branch 'main'.  
remote: Updating submodules.  
remote: Preparing deployment for commit id '25f18051e9'.  
remote: Generating deployment script.  
remote: Running deployment command...  
remote: Handling Basic Web Site deployment.  
remote: Kudu sync from: '/home/site/repository' to: '/home/site/wwwroot'  
remote: Copying file: '.gitignore'  
remote: Copying file: 'LICENSE'  
remote: Copying file: 'README.md'  
remote: Copying file: 'index.php'  
remote: Ignoring: .git  
remote: Finished successfully.  
remote: Running post deployment command(s)...  
remote: Deployment successful.  
To https://<app-name>.scm.azurewebsites.net/<app-name>.git  
cc39b1e..25f1805 main -> main
```

Browse to the app

Browse to the deployed application using your web browser.

```
http://<app-name>.azurewebsites.net
```

The PHP sample code is running in an Azure App Service web app.



Congratulations! You've deployed your first PHP app to App Service.

Update locally and redeploy the code

1. Using a local text editor, open the `index.php` file within the PHP app, and make a small change to the text within the string next to `echo`:

```
echo "Hello Azure!";
```

2. In the local terminal window, commit your changes in Git, and then push the code changes to Azure.

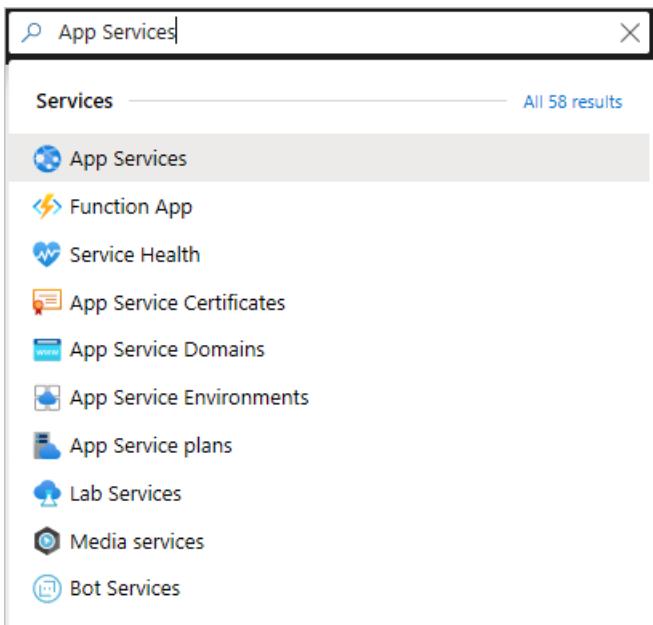
```
git commit -am "updated output"  
git push azure main
```

3. Once deployment has completed, return to the browser window that opened during the **Browse to the app** step, and refresh the page.



Manage your new Azure app

1. Go to the [Azure portal](#) to manage the web app you created. Search for and select **App Services**.



2. Select the name of your Azure app.

A screenshot of the Microsoft Azure portal showing the "App Services" blade. The left sidebar shows "App Services" under "Microsoft". The main area displays a table with one item: "php-docs-hello-world" in the NAME column, "myResourceGroup" in the RESOURCE GROUP column, "West Europe" in the LOCATION column, and "Standard: 1 Small" in the APP SERVICE PLAN column. The table has columns for NAME, RESOURCE GROUP, LOCATION, STATUS, and APP SERVICE PLAN.

Your web app's **Overview** page will be displayed. Here, you can perform basic management tasks like **Browse**, **Stop**, **Restart**, and **Delete**.

A screenshot of the Microsoft Azure portal showing the "Overview" page for the "php-docs-hello-world" web app. The left sidebar shows "php-docs-hello-world" under "App Services". The main area has a toolbar with "Browse", "Stop", "Swap", "Restart", "Delete", and "More". The "Essentials" section displays resource group ("myResourceGroup"), status ("Running"), location ("West Europe"), subscription name ("Field Engineer Demo"), and subscription ID ("5d6c94cd-6781-43e3-8a84-ceef4c28850e"). It also shows the URL ("http://php-docs-hello-world.azurewebsites...") and Git deployment details. The "Monitoring" section shows "Requests and errors".

The web app menu provides different options for configuring your app.

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

Next steps

[PHP with MySQL](#)

[Configure PHP app](#)

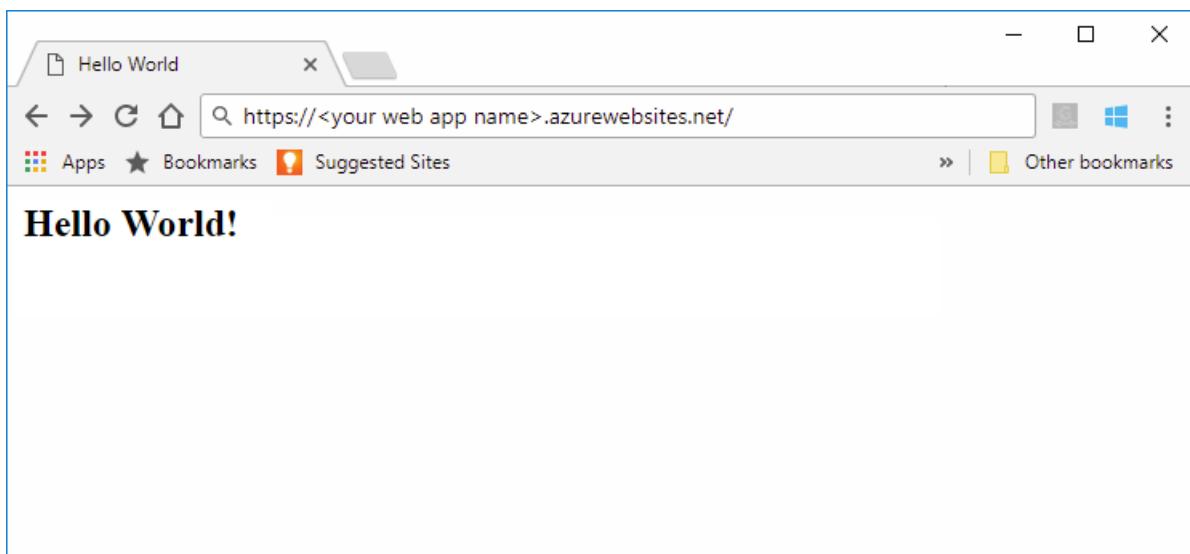
Quickstart: Create a Java app on Azure App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This quickstart shows how to use the Azure CLI with the Azure Web App Plugin for Maven to deploy a .jar file, or .war file. Use the tabs to switch between Java SE and Tomcat instructions.

If Maven isn't your preferred development tool, check out our similar tutorials for Java developers:

- [Gradle](#)
- [IntelliJ IDEA](#)
- [Eclipse](#)
- [Visual Studio Code](#)



If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	A screenshot of a code block. At the top right, there is a red-bordered 'Try It' button. To its left is the text 'Azure CLI'. Below the code block are 'Copy' and 'Try It' buttons.
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	A screenshot of a blue rectangular button with white text. On the left is a small icon of a terminal window. Next to it, the text 'Launch Cloud Shell' is written in white.

OPTION	EXAMPLE/LINK
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

Create a Java app

- [Java SE](#)
- [Tomcat](#)
- [JBoss EAP](#)

Clone the [Spring Boot Getting Started](#) sample project.

```
git clone https://github.com/spring-guides/gs-spring-boot
```

Change directory to the completed project.

```
cd gs-spring-boot/complete
```

Configure the Maven plugin

The deployment process to Azure App Service will use your Azure credentials from the Azure CLI automatically. If the Azure CLI is not installed locally, then the Maven plugin will authenticate with Oauth or device login. For more information, see [authentication with Maven plugins](#).

Run the Maven command below to configure the deployment. This command will help you to set up the App Service operating system, Java version, and Tomcat version.

```
mvn com.microsoft.azure:azure-webapp-maven-plugin:2.2.0:config
```

- [Java SE](#)
- [Tomcat](#)
- [JBoss EAP](#)

1. If prompted with **Subscription** option, select the proper `Subscription` by entering the number printed at the line start.
2. When prompted with **Web App** option, select the default option, `<create>`, by pressing enter.
3. When prompted with **OS** option, select **Windows** by entering `1`.
4. When prompted with **javaVersion** option, select **Java 8** by entering `1`.

5. When prompted with **Pricing Tier** option, select **P1v2** by entering **10**.

6. Finally, press enter on the last prompt to confirm your selections.

Your summary output will look similar to the snippet shown below.

```
Please confirm webapp properties
Subscription Id : ****_**_**_**_**_*****
AppName : spring-boot-1599007390755
ResourceGroup : spring-boot-1599007390755-rg
Region : centralus
PricingTier : P1v2
OS : Windows
Java : Java 8
Web server stack : Java SE
Deploy to slot : false
Confirm (Y/N)? : Y
[INFO] Saving configuration to pom.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 41.118 s
[INFO] Finished at: 2020-09-01T17:43:45-07:00
[INFO] -----
```

- [Java SE](#)
- [Tomcat](#)
- [JBoss EAP](#)

1. When prompted with **Subscription** option, select the proper **Subscription** by entering the number printed at the line start.

2. When prompted with **Web App** option, select the default option, **<create>**, by pressing enter.

3. When prompted with **OS** option, select **Linux** by pressing enter.

4. When prompted with **javaVersion** option, select **Java 8** by entering **1**.

5. When prompted with **Pricing Tier** option, select **P1v2** by entering **9**.

6. Finally, press enter on the last prompt to confirm your selections.

```
Please confirm webapp properties
Subscription Id : ****_**_**_**_**_*****
AppName : spring-boot-1599007116351
ResourceGroup : spring-boot-1599007116351-rg
Region : centralus
PricingTier : P1v2
OS : Linux
Web server stack : Java SE
Deploy to slot : false
Confirm (Y/N)? : Y
[INFO] Saving configuration to pom.
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 20.925 s
[INFO] Finished at: 2020-09-01T17:38:51-07:00
[INFO] -----
```

You can modify the configurations for App Service directly in your **pom.xml** if needed. Some common ones are

listed below:

PROPERTY	REQUIRED	DESCRIPTION	VERSION
<code><schemaVersion></code>	false	Specify the version of the configuration schema. Supported values are: <code>v1</code> , <code>v2</code> .	1.5.2
<code><subscriptionId></code>	false	Specify the subscription ID.	0.1.0+
<code><resourceGroup></code>	true	Azure Resource Group for your Web App.	0.1.0+
<code><appName></code>	true	The name of your Web App.	0.1.0+
<code><region></code>	false	Specifies the region where your Web App will be hosted; the default value is <code>centralus</code> . All valid regions at Supported Regions section.	0.1.0+
<code><pricingTier></code>	false	The pricing tier for your Web App. The default value is <code>P1v2</code> for production workload, while <code>B2</code> is the recommended minimum for Java dev/test. Learn more	0.1.0+
<code><runtime></code>	false	The runtime environment configuration, you could see the detail here .	0.1.0+
<code><deployment></code>	false	The deployment configuration, you could see the details here .	0.1.0+

Be careful about the values of `<appName>` and `<resourceGroup>` (`helloworld-1590394316693` and `helloworld-1590394316693-rg` accordingly in the demo), they will be used later.

I ran into an issue

Deploy the app

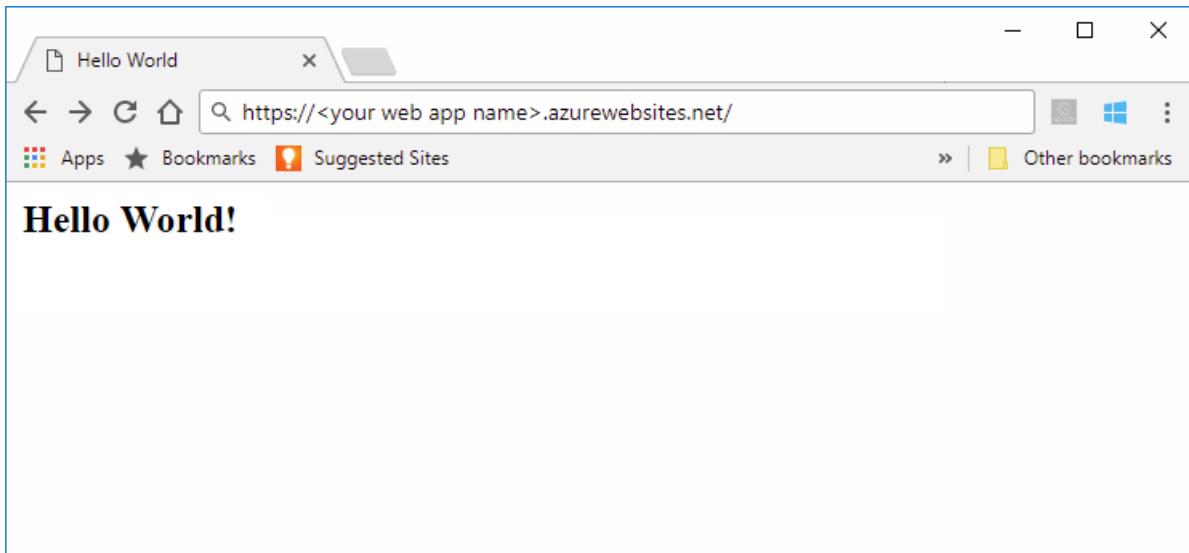
With all the configuration ready in your pom file, you can deploy your Java app to Azure with one single command.

```
mvn package azure-webapp:deploy
```

NOTE

For JBoss EAP, run `mvn package azure-webapp:deploy -DskipTests` to disable testing, as it requires Wildfly to be installed locally.

Once deployment has completed, your application will be ready at <http://<appName>.azurewebsites.net/> (<http://helloworld-1590394316693.azurewebsites.net> in the demo). Open the url with your local web browser, you should see



Congratulations! You've deployed your first Java app to App Service.

[I ran into an issue](#)

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group from portal, or by running the following command in the Cloud Shell:

```
az group delete --name <your resource group name; for example: helloworld-1558400876966-rg> --yes
```

This command may take a minute to run.

Next steps

[Connect to Azure DB for PostgreSQL with Java](#)

[Set up CI/CD](#)

[Pricing Information](#)

[Aggregate Logs and Metrics](#)

[Scale up](#)

[Azure for Java Developers Resources](#)

[Configure your Java app](#)

Quickstart: Create a Python app using Azure App Service on Linux

11/2/2021 • 8 minutes to read • [Edit Online](#)

In this quickstart, you deploy a Python web app to [App Service on Linux](#), Azure's highly scalable, self-patching web hosting service. You use the local [Azure command-line interface \(CLI\)](#) on a Mac, Linux, or Windows computer to deploy a sample with either the Flask or Django frameworks. The web app you configure uses a basic App Service tier that incurs a small cost in your Azure subscription.

Set up your initial environment

1. Have an Azure account with an active subscription. [Create an account for free](#).
2. Install [Python 3.6 or higher](#).
3. Install the [Azure CLI](#), with which you run commands in any shell to provision and configure Azure resources.

Open a terminal window and check your Python version is 3.6 or higher:

- [Bash](#)
- [PowerShell](#)
- [Cmd](#)

```
python3 --version
```

Check that your Azure CLI version is 2.0.80 or higher:

```
az --version
```

Then sign in to Azure through the CLI:

```
az login
```

This command opens a browser to gather your credentials. When the command finishes, it shows JSON output containing information about your subscriptions.

Once signed in, you can run Azure commands with the Azure CLI to work with resources in your subscription.

Having issues? [Let us know](#).

Clone the sample

Clone the sample repository using the following command and navigate into the sample folder. ([Install git](#) if you don't have git already.)

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

```
git clone https://github.com/Azure-Samples/python-docs-hello-django
```

The sample contains framework-specific code that Azure App Service recognizes when starting the app. For more information, see [Container startup process](#).

Having issues? [Let us know](#).

Run the sample

1. Navigate into in the `python-docs-hello-world` folder:

```
cd python-docs-hello-world
```

2. Create a virtual environment and install dependencies:

- [Bash](#)
- [PowerShell](#)
- [Cmd](#)

```
# Linux systems only
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

# Git Bash on Windows only
py -3 -m venv .venv
source .venv\scripts\activate
pip install -r requirements.txt
```

If you're on a Windows system and see the error "'source' is not recognized as an internal or external command," make sure you're either running in the Git Bash shell, or use the commands shown in the **Cmd** tab above.

If you encounter "[Errno 2] No such file or directory: 'requirements.txt'.", make sure you're in the `python-docs-hello-world` folder.

3. Run the development server.

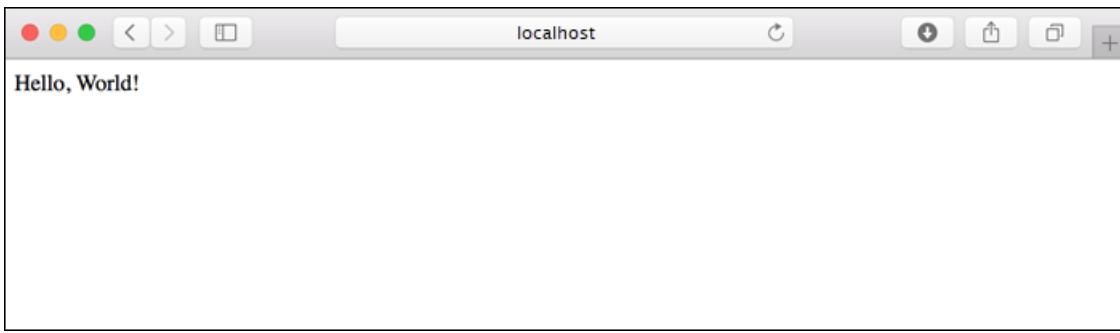
```
flask run
```

By default, the server assumes that the app's entry module is in `app.py`, as used in the sample.

If you use a different module name, set the `FLASK_APP` environment variable to that name.

If you encounter the error, "Could not locate a Flask application. You did not provide the 'FLASK_APP' environment variable, and a 'wsgi.py' or 'app.py' module was not found in the current directory.", make sure you're in the `python-docs-hello-world` folder that contains the sample.

4. Open a web browser and go to the sample app at `http://localhost:5000/`. The app displays the message **Hello, World!**.



5. In your terminal window, press **Ctrl+C** to exit the development server.

1. Navigate into the *python-docs-hello-django* folder:

```
cd python-docs-hello-django
```

2. Create a virtual environment and install dependencies:

- [Bash](#)
- [PowerShell](#)
- [Cmd](#)

```
# Linux systems only
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt

# Git Bash on Windows only
py -3 -m venv .venv
source .venv\\scripts\\activate
pip install -r requirements.txt
```

If you're on a Windows system and see the error "'source' is not recognized as an internal or external command," make sure you're either running in the Git Bash shell, or use the commands shown in the **Cmd** tab above.

If you encounter "[Errno 2] No such file or directory: 'requirements.txt'.", make sure you're in the *python-docs-hello-django* folder.

3. Run the development server.

```
python manage.py runserver
```

4. Open a web browser and go to the sample app at <http://localhost:8000/>. The app displays the message **Hello, World!**.



5. In your terminal window, press **Ctrl+C** to exit the development server.

Having issues? [Let us know](#).

Deploy the sample

Deploy the code in your local folder (*python-docs-hello-world*) using the `az webapp up` command:

```
az webapp up --sku B1 --name <app-name>
```

- If the `az` command isn't recognized, be sure you have the Azure CLI installed as described in [Set up your initial environment](#).
- If the `webapp` command isn't recognized, because that your Azure CLI version is 2.0.80 or higher. If not, [install the latest version](#).
- Replace `<app_name>` with a name that's unique across all of Azure (*valid characters are a-z, 0-9, and -*). A good pattern is to use a combination of your company name and an app identifier.
- The `--sku B1` argument creates the web app on the Basic pricing tier, which incurs a small hourly cost. Omit this argument to use a faster premium tier.
- You can optionally include the argument `--location <location-name>` where `<location_name>` is an available Azure region. You can retrieve a list of allowable regions for your Azure account by running the `az account list-locations` command.
- If you see the error, "Could not auto-detect the runtime stack of your app," make sure you're running the command in the *python-docs-hello-world* folder (Flask) or the *python-docs-hello-django* folder (Django) that contains the *requirements.txt* file. (See [Troubleshooting auto-detect issues with az webapp up](#) (GitHub).)

The command may take a few minutes to complete. While running, it provides messages about creating the resource group, the App Service plan and hosting app, configuring logging, then performing ZIP deployment. It then gives the message, "You can launch the app at <http://<app-name>.azurewebsites.net>", which is the app's URL on Azure.

```
Creating Resource group 'appsvc_rg_Linux_centralus' ...
Resource group creation complete
Creating App service plan 'appsvc_asp_Linux_centralus' ...
App service plan creation complete
Creating app 'msdocs-appservice-python12345' ....
Configuring default logging for the app, if not already enabled
Creating zip with contents of dir D:\Examples\python-docs-hello-world ...
Getting scm site credentials for zip deployment
Starting zip deployment. This operation can take a while to complete ...
Deployment endpoint responded with status code 202
You can launch the app at http://msdocs-appservice-python12345.azurewebsites.net
{
  "URL": "http://msdocs-appservice-python12345.net",
  "appserviceplan": "appsvc_asp_Linux_centralus",
  "location": "eastus",
  "name": "msdocs-appservice-python12345",
  "os": "Linux",
  "resourcegroup": "appsvc_rg_Linux_centralus",
  "runtime_version": "python|3.7",
  "runtime_version_detected": "-",
  "sku": "FREE",
  "src_path": "D:\\Examples\\python-docs-hello-world"
}
```

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

NOTE

The `az webapp up` command does the following actions:

- Create a default [resource group](#).
- Create a default [App Service plan](#).
- [Create an app](#) with the specified name.
- [Zip deploy](#) all files from the current working directory, [with build automation enabled](#).
- Cache the parameters locally in the `.azure/config` file so that you don't need to specify them again when deploying later with `az webapp up` or other Azure CLI commands. The cached values are used automatically by default.

Browse to the app

Browse to the deployed application in your web browser at the URL `http://<app-name>.azurewebsites.net`. It can take a minute or two for the the app to start, so if you see a default app page, wait a minute and refresh the browser.

The Python sample code is running a Linux container in App Service using a built-in image.



Congratulations! You've deployed your Python app to App Service.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

Redeploy updates

In this section, you make a small code change and then redeploy the code to Azure. The code change includes a `print` statement to generate logging output that you work with in the next section.

Open `app.py` in an editor and update the `hello` function to match the following code.

```
def hello():
    print("Handling request to home page.")
    return "Hello, Azure!"
```

Open `hello/views.py` in an editor and update the `hello` function to match the following code.

```
def hello(request):
    print("Handling request to home page.")
    return HttpResponse("Hello, Azure!")
```

Save your changes, then redeploy the app using the `az webapp up` command again:

```
az webapp up
```

This command uses values that are cached locally in the `.azure/config` file, including the app name, resource group, and App Service plan.

Once deployment is complete, switch back to the browser window open to <http://<app-name>.azurewebsites.net>. Refresh the page, which should display the modified message:



Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

TIP

Visual Studio Code provides powerful extensions for Python and Azure App Service, which simplify the process of deploying Python web apps to App Service. For more information, see [Deploy Python apps to App Service from Visual Studio Code](#).

Stream logs

You can access the console logs generated from inside the app and the container in which it runs. Logs include any output generated using `print` statements.

To stream logs, run the `az webapp log tail` command:

```
az webapp log tail
```

You can also include the `--logs` parameter with the `az webapp up` command to automatically open the log stream on deployment.

Refresh the app in the browser to generate console logs, which include messages describing HTTP requests to the app. If no output appears immediately, try again in 30 seconds.

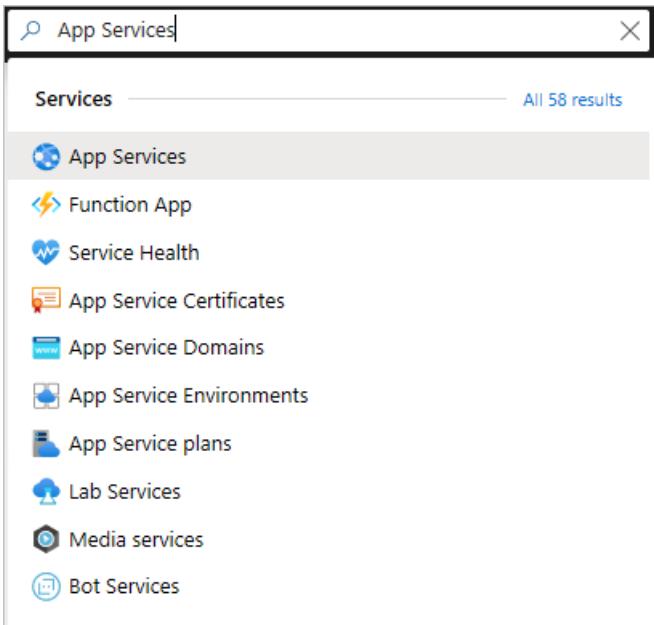
You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, press **Ctrl+C** in the terminal.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

Manage the Azure app

Go to the [Azure portal](#) to manage the app you created. Search for and select **App Services**.



Select the name of your Azure app.

A screenshot of the Azure portal's "App Services" blade. The URL in the address bar is "Home > App Services". The page title is "App Services Microsoft". There are buttons for "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Stop", and "Delete". A message says "Subscriptions: All 2 selected - Don't see a subscription? Open Directory + Subscription settings". There are filters for "Filter by name...", "All subscriptions", "All resource groups", "All locations", "All tags", and "No grouping". A table lists 7 items: one row is collapsed, and one row is expanded, showing details like Name (<app name>), Status (Running), App Type (Web App), App Service Plan (user_asp_Linux_centralus_0), Location (Central US), and Subscription (SomeSubscriptionName). The row for the expanded app has a red border around the "Name" column.

Selecting the app opens its **Overview** page, where you can perform basic management tasks like browse, stop, start, restart, and delete.

A screenshot of the Azure portal's "Overview" page for a specific app service. The URL is "Home > <App Name> App Service". The page title is "<App Name> App Service". On the left is a navigation menu with links: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots), and Deployment Center. The main content area shows the app's configuration: Resource group (user_rg_Linux_centralus), Status (Running), Location (Central US), Subscription (SomeSubscriptionName), Subscription ID (00000000-0000-0000-0000-000000000000), and Tags (cli : webapp_up). Action buttons include Browse, Stop, Swap, Restart, Delete, Get publish profile, and Reset publish profile.

The App Service menu provides different pages for configuring your app.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

Clean up resources

In the preceding steps, you created Azure resources in a resource group. The resource group has a name like "appsvc_rg_Linux_CentralUS" depending on your location. If you keep the web app running, you will incur some ongoing costs (see [App Service pricing](#)).

If you don't expect to need these resources in the future, delete the resource group by running the following command:

```
az group delete --no-wait
```

The command uses the resource group name cached in the `.azure/config` file.

The `--no-wait` argument allows the command to return before the operation is complete.

Having issues? [Let us know](#).

Next steps

[Tutorial: Python \(Django\) web app with PostgreSQL](#)

[Configure Python app](#)

[Add user sign-in to a Python web app](#)

[Tutorial: Run Python app in custom container](#)

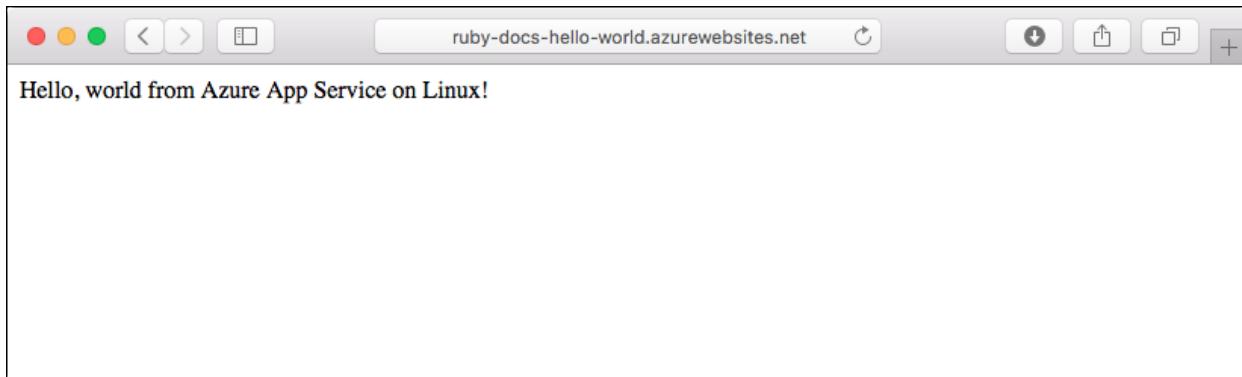
Create a Ruby on Rails App in App Service

11/2/2021 • 6 minutes to read • [Edit Online](#)

Azure App Service on Linux provides a highly scalable, self-patching web hosting service using the Linux operating system. This quickstart tutorial shows how to deploy a Ruby on Rails app to App Service on Linux using the Cloud Shell.

NOTE

The Ruby development stack only supports Ruby on Rails at this time. If you want to use a different platform, such as Sinatra, or if you want to use an unsupported Ruby version, you need to [run it in a custom container](#).



If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Prerequisites

- [Install Ruby 2.6 or higher](#)
- [Install Git](#)

Download the sample

1. In a terminal window, clone the sample application to your local machine, and navigate to the directory containing the sample code.

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

2. Make sure the default branch is `main`.

```
git branch -m main
```

TIP

The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main`, this tutorial also shows you how to deploy a repository from `main`. For more information, see [Change deployment branch](#).

Run the application locally

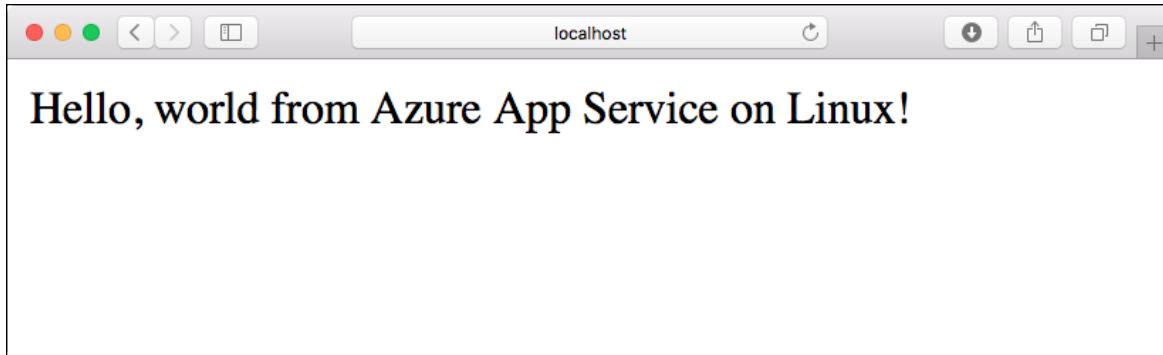
1. Install the required gems. There's a `Gemfile` included in the sample, so just run the following command:

```
bundle install
```

2. Once the gems are installed, start the app:

```
bundle exec rails server
```

3. Using your web browser, navigate to `http://localhost:3000` to test the app locally.



Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	A screenshot of a code block. In the top right corner, there is a green button labeled "Try It". This button is highlighted with a red box.
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	A screenshot of a blue button labeled "Launch Cloud Shell". This button is highlighted with a red box.
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal .	A screenshot of a blue menu bar. On the far left, there is a white square icon with a black sigma symbol. This icon is highlighted with a red box.

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the `az webapp deployment user set` command in Azure Cloud Shell. Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

Create a resource group

A **resource group** is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service on Linux in **Basic** tier, run the `az appservice list-locations --sku B1 --linux-workers-enabled` command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

Create an Azure App Service plan

In the Cloud Shell, create an App Service plan with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{  
    "freeOfferExpirationTime": null,  
    "geoRegion": "West Europe",  
    "hostingEnvironmentProfile": null,  
    "id": "/subscriptions/0000-  
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",  
    "kind": "linux",  
    "location": "West Europe",  
    "maximumNumberOfWorkers": 1,  
    "name": "myAppServicePlan",  
    < JSON data removed for brevity. >  
    "targetWorkerSizeId": 0,  
    "type": "Microsoft.Web/serverfarms",  
    "workerTierName": null  
}
```

Create a web app

1. Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `RUBY|2.6`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime  
'RUBY|2.6' --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-  
name>.scm.azurewebsites.net/<app-name>.git'  
{  
    "availabilityState": "Normal",  
    "clientAffinityEnabled": true,  
    "clientCertEnabled": false,  
    "cloningInfo": null,  
    "containerSize": 0,  
    "dailyMemoryTimeQuota": 0,  
    "defaultHostName": "<app-name>.azurewebsites.net",  
    "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-  
name>.git",  
    "enabled": true,  
    < JSON data removed for brevity. >  
}
```

You've created an empty new web app, with git deployment enabled.

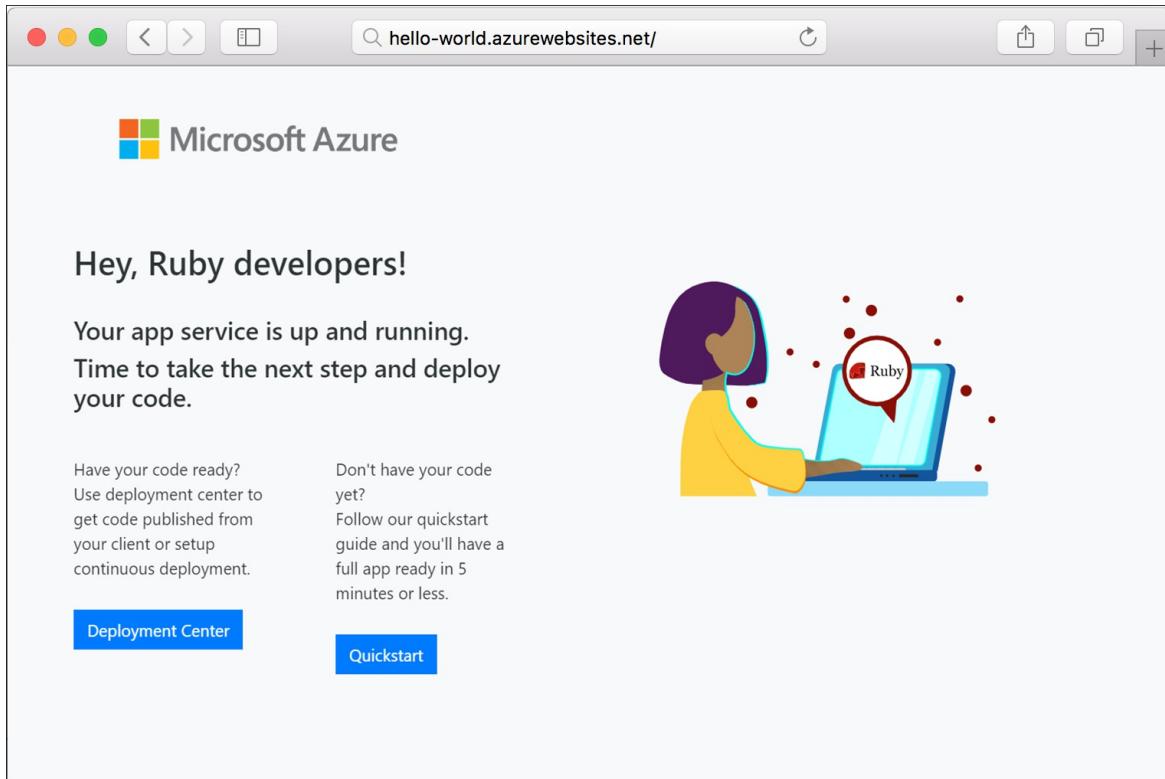
NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format `https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

2. Browse to the app to see your newly created web app with built-in image. Replace `<app-name>` with your web app name.

```
http://<app_name>.azurewebsites.net
```

Here is what your new web app should look like:



Deploy your application

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace `<deploymentLocalGitUrl-from-create-step>` with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in [Configure a deployment user](#), not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

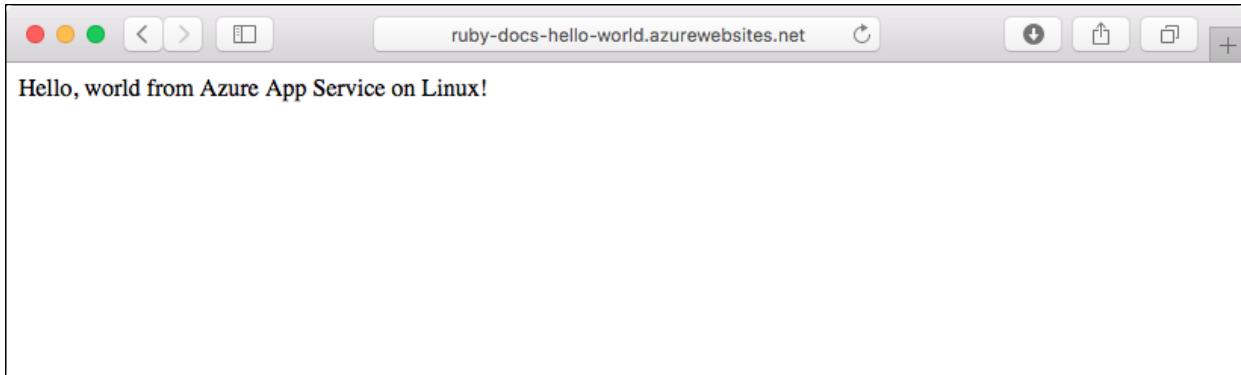
This command may take a few minutes to run. While running, it displays information similar to the following example:

```
remote: Using turbolinks 5.2.0
remote: Using uglifier 4.1.20
remote: Using web-console 3.7.0
remote: Bundle complete! 18 Gemfile dependencies, 78 gems now installed.
remote: Bundled gems are installed into `'/tmp/bundle'`
remote: Zipping up bundle contents
remote: .....
remote: ~/site/repository
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Deployment successful.
remote: App container will begin restart within 10 seconds.
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
 a6e73a2..ae34be9  main -> main
```

Browse to the app

Once the deployment has completed, wait about 10 seconds for the web app to restart, and then navigate to the web app and verify the results.

<http://<app-name>.azurewebsites.net>



NOTE

While the app is restarting, you may observe the HTTP status code `Error 503 Server unavailable` in the browser, or the `Hey, Ruby developers!` default page. It may take a few minutes for the app to fully restart.

Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

```
az group delete --name myResourceGroup
```

Next steps

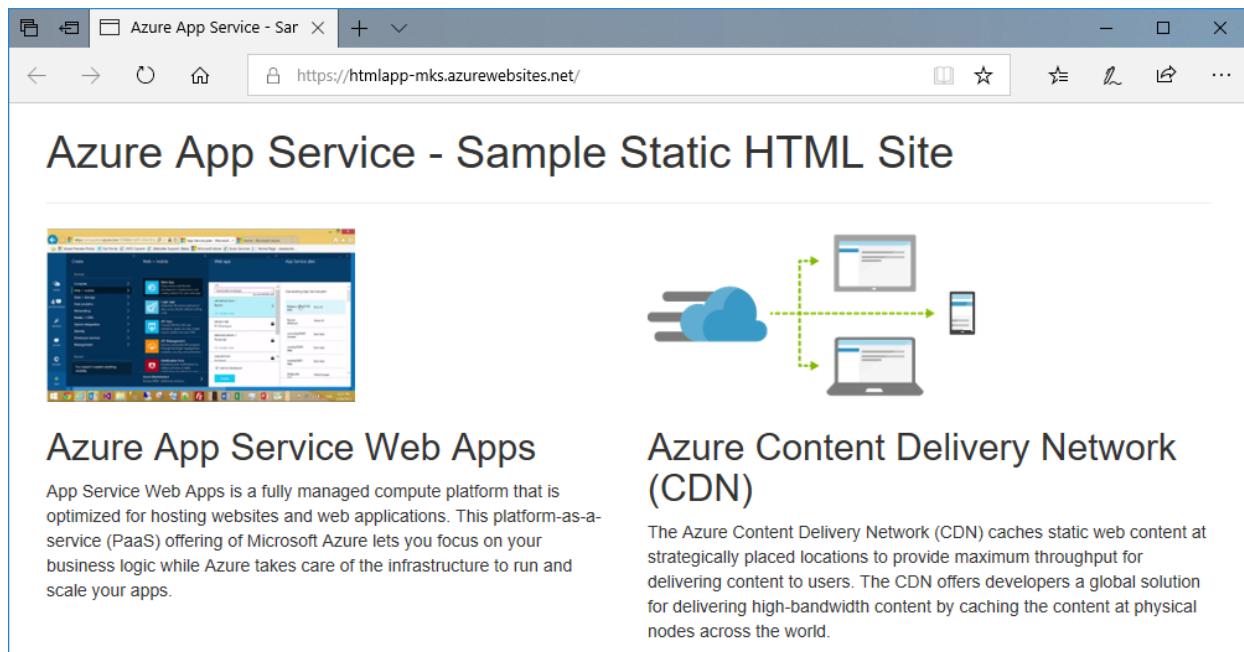
[Tutorial: Ruby on Rails with Postgres](#)

[Configure Ruby app](#)

Create a static HTML web app in Azure

11/2/2021 • 3 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy a basic HTML+CSS site to Azure App Service. You'll complete this quickstart in [Cloud Shell](#), but you can also run these commands locally with [Azure CLI](#).



Azure App Service Web Apps

App Service Web Apps is a fully managed compute platform that is optimized for hosting websites and web applications. This platform-as-a-service (PaaS) offering of Microsoft Azure lets you focus on your business logic while Azure takes care of the infrastructure to run and scale your apps.

Azure Content Delivery Network (CDN)

The Azure Content Delivery Network (CDN) caches static web content at strategically placed locations to provide maximum throughput for delivering content to users. The CDN offers developers a global solution for delivering high-bandwidth content by caching the content at physical nodes across the world.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to https://shell.azure.com , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.

2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

Download the sample

In the Cloud Shell, create a quickstart directory and then change to it.

```
mkdir quickstart  
cd $HOME/quickstart
```

Next, run the following command to clone the sample app repository to your quickstart directory.

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

Create a web app

Change to the directory that contains the sample code and run the `az webapp up` command. In the following example, replace `<app_name>` with a unique app name. Static content is indicated by the `--html` flag.

```
cd html-docs-hello-world  
az webapp up --location westeurope --name <app_name> --html
```

The `az webapp up` command does the following actions:

- Create a default resource group.
- 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.

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

```
{  
  "app_url": "https://<app_name>.azurewebsites.net",  
  "location": "westerneurope",  
  "name": "<app_name>",  
  "os": "Windows",  
  "resourcegroup": "appsvc_rg_Windows_westerneurope",  
  "serverfarm": "appsvc_asp_Windows_westerneurope",  
  "sku": "FREE",  
  "src_path": "/home/<username>/quickstart/html-docs-hello-world ",  
  &lt; JSON data removed for brevity. &gt;  
}
```

Make a note of the `resourceGroup` value. You need it for the [clean up resources](#) section.

Browse to the app

In a browser, go to the app URL: `http://<app_name>.azurewebsites.net`.

The page is running as an Azure App Service web app.

Azure App Service - Sample Static HTML Site

Azure App Service Web Apps

App Service Web Apps is a fully managed compute platform that is optimized for hosting websites and web applications. This platform-as-a-service (PaaS) offering of Microsoft Azure lets you focus on your business logic while Azure takes care of the infrastructure to run and scale your apps.

Azure Content Delivery Network (CDN)

The Azure Content Delivery Network (CDN) caches static web content at strategically placed locations to provide maximum throughput for delivering content to users. The CDN offers developers a global solution for delivering high-bandwidth content by caching the content at physical nodes across the world.

Congratulations! You've deployed your first HTML app to App Service.

Update and redeploy the app

In the Cloud Shell, type `nano index.html` to open the nano text editor. In the `<h1>` heading tag, change "Azure App Service - Sample Static HTML Site" to "Azure App Service", as shown below.

```
Bash | ⌁ ? ⌂ ↑ ⌄ *** File: index.html Modified
GNU nano 2.5.3
<title>Azure App Service - Sample Static HTML Site</title>
<!-- Bootstrap core CSS -->
<link href="css/bootstrap.min.css" rel="stylesheet">
</head>

<body>
  <div class="navbar-wrapper">
    <div class="container">
      <h1>Azure App Service</h1>
      <hr/>
    </div>
  </div>

  <!-- Wrap the rest of the page in another container to center all the content. -->

  <div class="container">
    ^G Get Help    ^O Write Out    ^W Where Is    ^K Cut Text    ^J Justify    ^C Cur Pos    ^Y Prev Page
    ^X Exit       ^R Read File   ^M Replace     ^U Uncut Text   ^T To Spell   ^G Go To Line  ^V Next Page

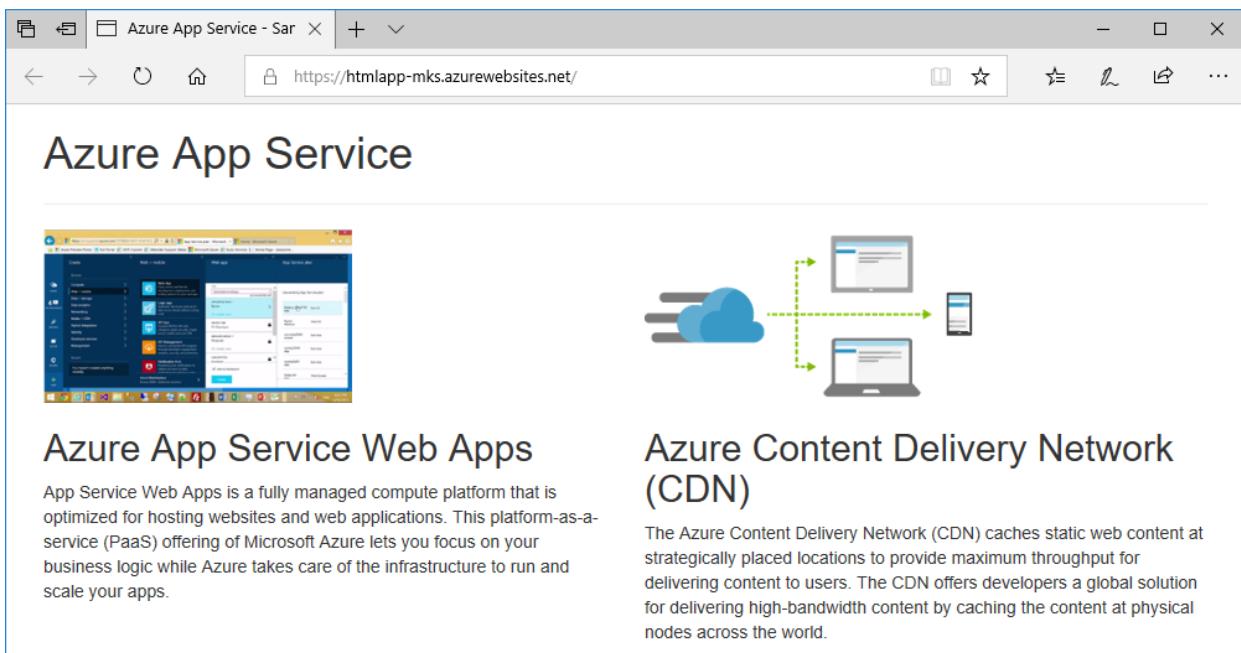
```

Save your changes and exit nano. Use the command `^o` to save and `^x` to exit.

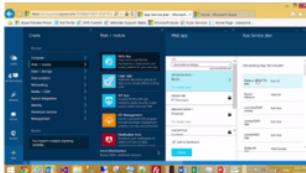
You'll now redeploy the app with the same `az webapp up` command.

```
az webapp up --location westeurope --name <app_name> --html
```

Once deployment has completed, switch back to the browser window that opened in the **Browse to the app** step, and refresh the page.



Azure App Service





Azure App Service Web Apps

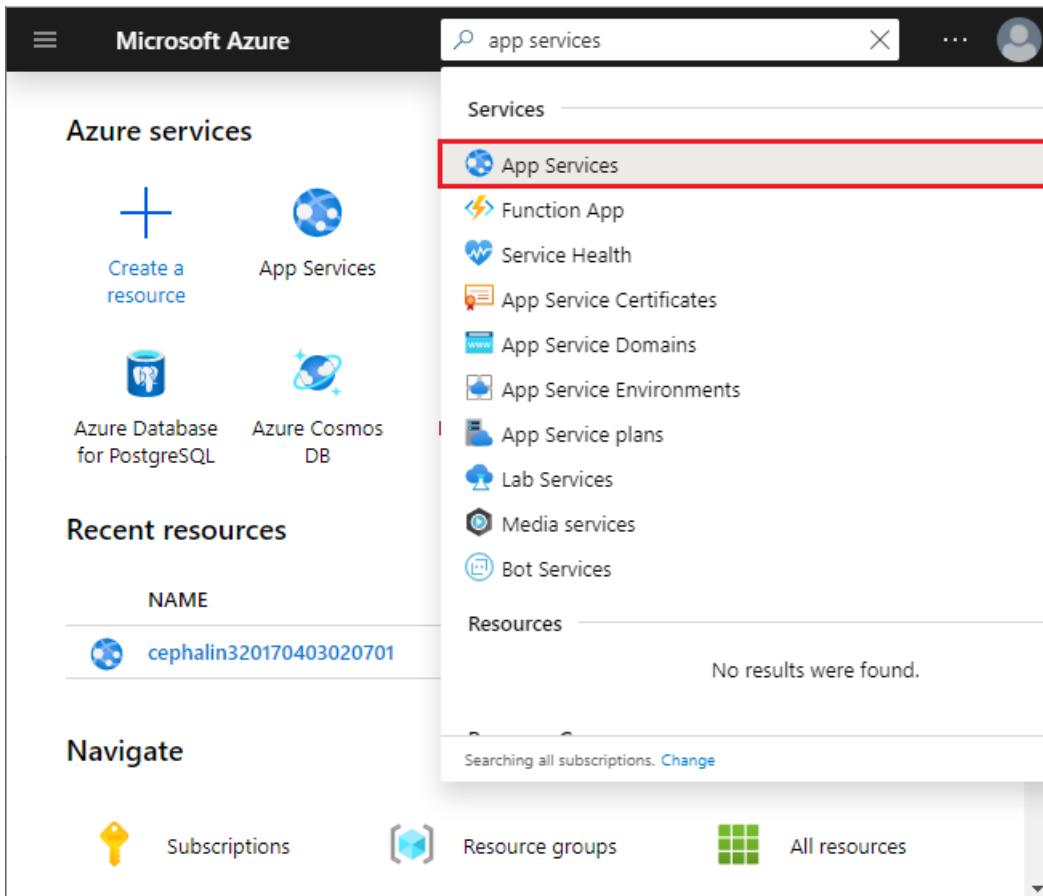
App Service Web Apps is a fully managed compute platform that is optimized for hosting websites and web applications. This platform-as-a-service (PaaS) offering of Microsoft Azure lets you focus on your business logic while Azure takes care of the infrastructure to run and scale your apps.

Azure Content Delivery Network (CDN)

The Azure Content Delivery Network (CDN) caches static web content at strategically placed locations to provide maximum throughput for delivering content to users. The CDN offers developers a global solution for delivering high-bandwidth content by caching the content at physical nodes across the world.

Manage your new Azure app

To manage the web app you created, in the [Azure portal](#), search for and select App Services.



Microsoft Azure

app services

Azure services

Create a resource

App Services

Azure Database for PostgreSQL

Azure Cosmos DB

Recent resources

NAME

cephalin320170403020701

App Services

Function App

Service Health

App Service Certificates

App Service Domains

App Service Environments

App Service plans

Lab Services

Media services

Bot Services

No results were found.

Searching all subscriptions. [Change](#)

Subscriptions

Resource groups

All resources

On the App Services page, select the name of your Azure app.

Subscriptions: All 2 selected – Don't see a subscription? Open Directory + Subscription settings

Name	Status	App Type	App Service Plan	Location
cephalin320170403020701	Running	Web App	test-sku	Central US
denniseastusbot	Running	Web App	z76-z763if-sp	East US
htmlapp-mks	Running	Web App	ServicePlane917530d...	Central US
myFirstAzureWebApp20190...	Running	Web App	ServicePlane917530d...	Central US
myfunctionapp04	Running	Function App	ASP-TimRG01-a65e	East US 2

You see your web app's Overview page. Here, you can perform basic management tasks like browse, stop, start, restart, and delete.

htmlapp-mks App Service

Resource group (change) : azscreens

Status	: Running	URL	: https://htmlapp-mks.azurewebsites.net
Location	: Central US	App Service Plan	: test-sku (B1: 1)
Subscription (change)	: A Subscription	FTP/deployment userna...	: htmlapp-mks\Deploy1234567
Subscription ID	: 12345678-1234-1234-1234-123456781234	FTP hostname	: ftp://waws-prod-dm1-145.ftp.azurewebsites.windows...
Tags (change)	: Click here to add tags	FTPS hostname	: ftps://waws-prod-dm1-145.ftp.azurewebsites.windows...

Diagnose and solve problems
Our self-service diagnostic and troubleshooting experience helps you identify and resolve issues with your web app.

Application Insights
Application Insights helps you detect and diagnose quality issues in your apps, and helps you understand what your users actually do with it.

App Service Advisor
App Service Advisor provides insights for improving app experience on the App Service platform. Recommendations are sorted by freshness, priority and impact to your app.

Http 5xx

Data In

Data Out

The left menu provides different pages for configuring your app.

Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell. Remember that the resource group name was automatically generated for you in the [create a web app](#) step.

```
az group delete --name appsvc_rg_Windows_westeurope
```

This command may take a minute to run.

Next steps

[Map custom domain](#)

Quickstart: Create App Service app using an ARM template

11/2/2021 • 5 minutes to read • [Edit Online](#)

Get started with [Azure App Service](#) by deploying a app to the cloud using an Azure Resource Manager template (ARM template) and [Azure CLI](#) in Cloud Shell. Because you use a free App Service tier, you incur no costs to complete this quickstart.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

Use the following button to deploy on **Linux**:



Use the following button to deploy on **Windows**:



Prerequisites

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#). It deploys an App Service plan and an App Service app on Windows. It's compatible with .NET Core, .NET Framework, PHP, Node.js, and Static HTML apps. For Java, see [Create Java app](#).

```
{  
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",  
  "parameters": {  
    "webAppName": {  
      "type": "string",  
      "defaultValue": "[concat('webApp-', uniqueString(resourceGroup().id))]",  
      "minLength": 2,  
      "metadata": {  
        "description": "Web app name."  
      }  
    },  
    "location": {  
      "type": "string",  
      "defaultValue": "[resourceGroup().location]",  
      "metadata": {  
        "description": "Location for all resources."  
      }  
    },  
    "sku": {  
      "type": "string",  
      "defaultValue": "F1".  
    }  
  },  
  "variables": {},  
  "resources": [  
    {  
      "type": "Microsoft.Web/sites",  
      "name": "[parameters('webAppName')]",  
      "apiVersion": "2019-02-01",  
      "location": "[parameters('location')]",  
      "sku": {  
        "name": "F1",  
        "tier": "Free",  
        "capacity": 1  
      },  
      "properties": {  
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', 'appserviceplan')]"  
      }  
    },  
    {  
      "type": "Microsoft.Web/serverfarms",  
      "name": "appserviceplan",  
      "apiVersion": "2019-02-01",  
      "location": "[parameters('location')]",  
      "sku": {  
        "name": "F1",  
        "tier": "Free",  
        "capacity": 1  
      },  
      "properties": {}  
    }  
  ]  
}
```

```
        },
        "metadata": {
            "description": "The SKU of App Service Plan."
        }
    },
    "language": {
        "type": "string",
        "defaultValue": ".net",
        "allowedValues": [
            ".net",
            "php",
            "node",
            "html"
        ],
        "metadata": {
            "description": "The language stack of the app."
        }
    },
    "helloWorld": {
        "type": "bool",
        "defaultValue": false,
        "metadata": {
            "description": "true = deploy a sample Hello World app."
        }
    },
    "repoUrl": {
        "type": "string",
        "defaultValue": "",
        "metadata": {
            "description": "Optional Git Repo URL"
        }
    }
},
"variables": {
    "appServicePlanPortalName": "[concat('AppServicePlan-', parameters('webAppName'))]",
    "gitRepoReference": {
        ".net": "https://github.com/Azure-Samples/app-service-web-dotnet-get-started",
        "node": "https://github.com/Azure-Samples/nodejs-docs-hello-world",
        "php": "https://github.com/Azure-Samples/php-docs-hello-world",
        "html": "https://github.com/Azure-Samples/html-docs-hello-world"
    },
    "gitRepoUrl": "[if(bool(parameters('helloWorld')), variables('gitRepoReference')[toLowerCase(parameters('language'))]), parameters('repoUrl'))]",
    "configReference": {
        ".net": {
            "comments": ".Net app. No additional configuration needed."
        },
        "html": {
            "comments": "HTML app. No additional configuration needed."
        },
        "php": {
            "phpVersion": "7.4"
        },
        "node": {
            "appSettings": [
                {
                    "name": "WEBSITE_NODE_DEFAULT_VERSION",
                    "value": "12.15.0"
                }
            ]
        }
    }
},
"resources": [
{
    "type": "Microsoft.Web/serverfarms",
    "apiVersion": "2020-06-01",
    "name": "[variables('appServicePlanPortalName')]",
    "location": "[parameters('location')]",
    "sku": {
        "tier": "Standard",
        "size": "S1",
        "family": "Standard"
    }
}
```

```

    "sku": {
      "name": "[parameters('sku')]"
    },
    {
      "type": "Microsoft.Web/sites",
      "apiVersion": "2020-06-01",
      "name": "[parameters('webAppName')]",
      "location": "[parameters('location')]",
      "dependsOn": [
        "[resourceId('Microsoft.Web/serverfarms', variables('appServicePlanPortalName'))]"
      ],
      "properties": {
        "siteConfig": "[variables('configReference')[parameters('language')]]",
        "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('appServicePlanPortalName'))]"
      },
      "resources": [
        {
          "condition": "[contains(variables('gitRepoUrl'), 'http')]",
          "type": "sourcecontrols",
          "apiVersion": "2020-06-01",
          "name": "web",
          "location": "[parameters('location')]",
          "dependsOn": [
            "[resourceId('Microsoft.Web/sites', parameters('webAppName'))]"
          ],
          "properties": {
            "repoUrl": "[variables('gitRepoUrl')]",
            "branch": "master",
            "isManualIntegration": true
          }
        }
      ]
    }
  ]
}

```

Two Azure resources are defined in the template:

- **Microsoft.Web/serverfarms**: create an App Service plan.
- **Microsoft.Web/sites**: create an App Service app.

This template contains several parameters that are predefined for your convenience. See the table below for parameter defaults and their descriptions:

PARAMETERS	TYPE	DEFAULT VALUE	DESCRIPTION
webAppName	string	"webApp- <uniqueString>"	App name
location	string	"[resourceGroup().location]"	App region
sku	string	"F1"	Instance size (F1 = Free Tier)
language	string	".net"	Programming language stack (.net, php, node, html)
helloWorld	boolean	False	True = Deploy "Hello World" app
repoUrl	string	" "	External Git repo (optional)

The template used in this quickstart is from [Azure Quickstart Templates](#). It deploys an App Service plan and an App Service app on Linux. It's compatible with all supported programming languages on App Service.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "webAppName": {
      "type": "string",
      "defaultValue": "[concat('webApp-', uniqueString(resourceGroup().id))]",
      "minLength": 2,
      "metadata": {
        "description": "Web app name."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for all resources."
      }
    },
    "sku": {
      "type": "string",
      "defaultValue": "F1",
      "metadata": {
        "description": "The SKU of App Service Plan."
      }
    },
    "linuxFxVersion": {
      "type": "string",
      "defaultValue": "DOTNETCORE|3.0",
      "metadata": {
        "description": "The Runtime stack of current web app"
      }
    },
    "repoUrl": {
      "type": "string",
      "defaultValue": " ",
      "metadata": {
        "description": "Optional Git Repo URL"
      }
    }
  },
  "variables": {
    "appServicePlanPortalName": "[concat('AppServicePlan-', parameters('webAppName'))]"
  },
  "resources": [
    {
      "type": "Microsoft.Web/serverfarms",
      "apiVersion": "2020-06-01",
      "name": "[variables('appServicePlanPortalName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "[parameters('sku')]"
      },
      "kind": "linux",
      "properties": {
        "reserved": true
      }
    },
    {
      "type": "Microsoft.Web/sites",
      "apiVersion": "2020-06-01",
      "name": "[parameters('webAppName')]",
      "location": "[parameters('location')]"
    }
  ]
}
```

```

  "dependsOn": [
    "[resourceId('Microsoft.Web/serverfarms', variables('appServicePlanPortalName'))]"
  ],
  "properties": {
    "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('appServicePlanPortalName'))]",
    "siteConfig": {
      "linuxFxVersion": "[parameters('linuxFxVersion')]"
    },
    "resources": [
      {
        "condition": "[contains(parameters('repoUrl'), 'http')]",
        "type": "sourcecontrols",
        "apiVersion": "2020-06-01",
        "name": "web",
        "location": "[parameters('location')]",
        "dependsOn": [
          "[resourceId('Microsoft.Web/sites', parameters('webAppName'))]"
        ],
        "properties": {
          "repoUrl": "[parameters('repoUrl')]",
          "branch": "master",
          "isManualIntegration": true
        }
      }
    ]
  }
]
}

```

Two Azure resources are defined in the template:

- **Microsoft.Web/serverfarms**: create an App Service plan.
- **Microsoft.Web/sites**: create an App Service app.

This template contains several parameters that are predefined for your convenience. See the table below for parameter defaults and their descriptions:

PARAMETERS	TYPE	DEFAULT VALUE	DESCRIPTION
webAppName	string	"webApp- <uniqueString>"	App name
location	string	"[resourceGroup().location]"	App region
sku	string	"F1"	Instance size (F1 = Free Tier)
linuxFxVersion	string	"DOTNETCORE 3.0"	"Programming language stack Version"
repoUrl	string	" "	External Git repo (optional)

Deploy the template

Azure CLI is used here to deploy the template. You can also use the Azure portal, Azure PowerShell, and REST API. To learn other deployment methods, see [Deploy templates](#).

The following code creates a resource group, an App Service plan, and a web app. A default resource group, App Service plan, and location have been set for you. Replace `<app-name>` with a globally unique app name (valid

characters are `a-z`, `0-9`, and `-`).

Run the code below to deploy a .NET framework app on Windows.

```
az group create --name myResourceGroup --location "southcentralus" &&
az deployment group create --resource-group myResourceGroup \
--parameters language=".net" helloWorld="true" webAppName=<app-name> \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.web/app-service-windows/azuredeploy.json"
```

Run the code below to create a Python app on Linux.

```
az group create --name myResourceGroup --location "southcentralus" &&
az deployment group create --resource-group myResourceGroup --parameters webAppName=<app-name>
linuxFxVersion="PYTHON|3.7" \
--template-uri "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.web/app-service-linux/azuredeploy.json"
```

To deploy a different language stack, update `linuxFxVersion` with appropriate values. Samples are shown below.

To show current versions, run the following command in the Cloud Shell:

```
az webapp config show --resource-group myResourceGroup --name <app-name> --query linuxFxVersion
```

LANGUAGE	EXAMPLE
.NET	linuxFxVersion="DOTNETCORE 3.0"
PHP	linuxFxVersion="PHP 7.4"
Node.js	linuxFxVersion="NODE 10.15"
Java	linuxFxVersion="JAVA 1.8 TOMCAT 9.0"
Python	linuxFxVersion="PYTHON 3.7"
Ruby	linuxFxVersion="RUBY 2.6"

NOTE

You can find more [Azure App Service template samples here](#).

Validate the deployment

Browse to `http://<app_name>.azurewebsites.net/` and verify it's been created.

Clean up resources

When no longer needed, [delete the resource group](#).

Next steps

[Deploy from local Git](#)

[ASP.NET Core with SQL Database](#)

[Python with Postgres](#)

[PHP with MySQL](#)

[Connect to Azure SQL database with Java](#)

[Map custom domain](#)

Run a custom container in Azure

11/2/2021 • 9 minutes to read • [Edit Online](#)

Azure App Service provides pre-defined application stacks on Windows like ASP.NET or Node.js, running on IIS. However, the preconfigured application stacks [lock down the operating system and prevent low-level access](#). Custom Windows containers do not have these restrictions, and let developers fully customize the containers and give containerized applications full access to Windows functionality.

This quickstart shows how to deploy an ASP.NET app, in a Windows image, to [Docker Hub](#) from Visual Studio. You run the app in a custom container in Azure App Service.

Prerequisites

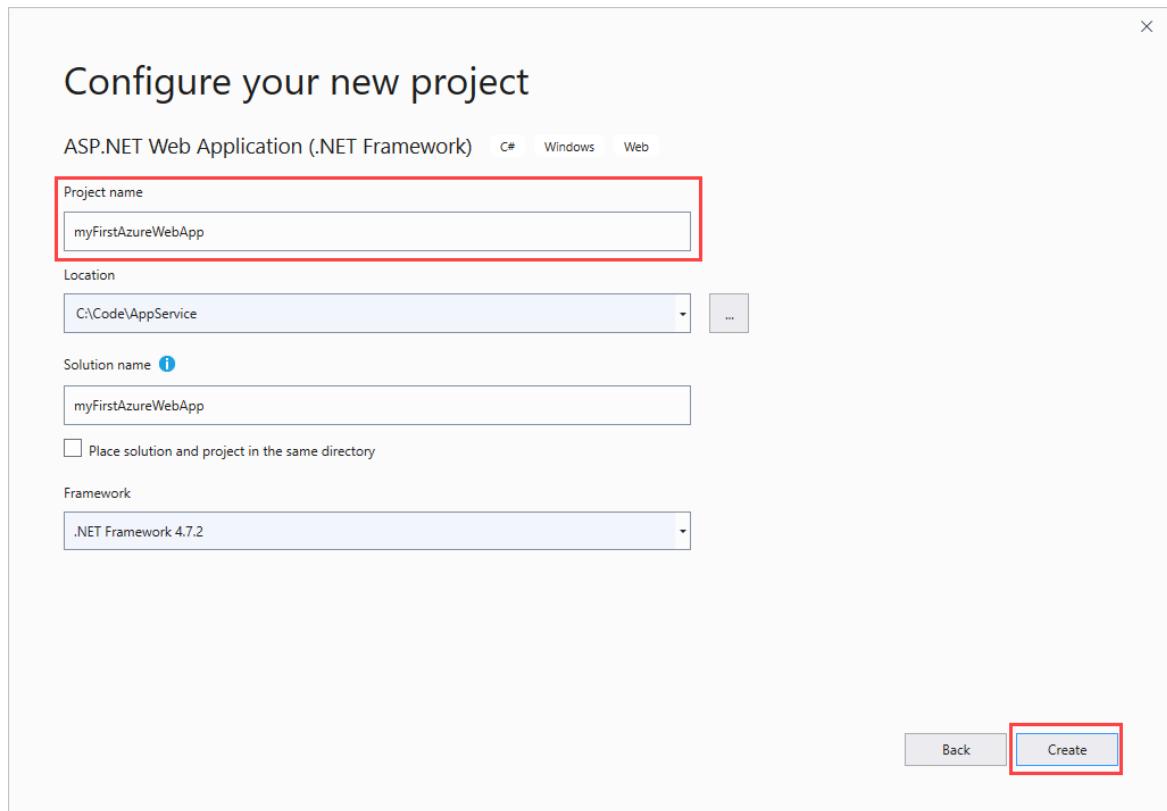
To complete this tutorial:

- [Sign up for a Docker Hub account](#)
- [Install Docker for Windows](#).
- [Switch Docker to run Windows containers](#).
- [Install Visual Studio 2019 with the ASP.NET and web development and Azure development workloads](#). If you've installed Visual Studio 2019 already:
 - Install the latest updates in Visual Studio by selecting **Help > Check for Updates**.
 - Add the workloads in Visual Studio by selecting **Tools > Get Tools and Features**.

Create an ASP.NET web app

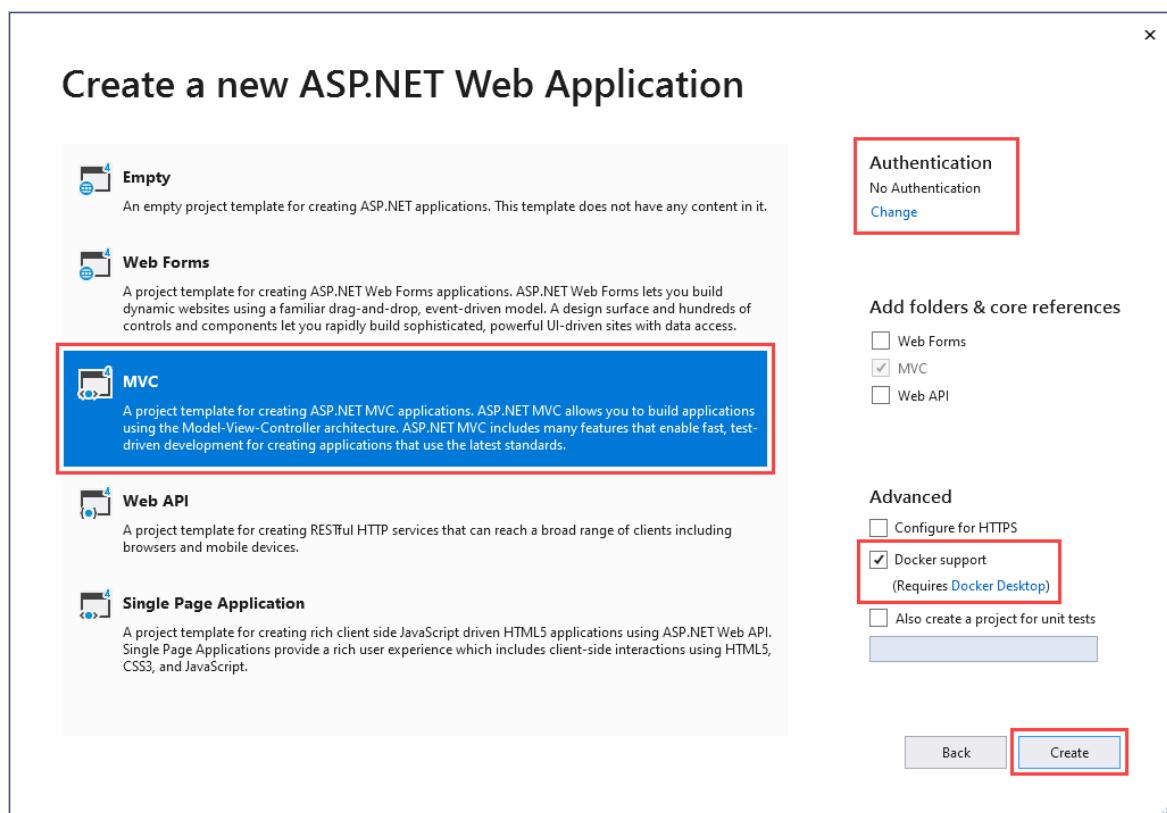
Create an ASP.NET web app by following these steps:

1. Open Visual Studio and then select **Create a new project**.
2. In **Create a new project**, find and choose **ASP.NET Web Application (.NET Framework)** for C#, then select **Next**.
3. In **Configure your new project**, name the application *myfirstazurewebapp*, and then select **Create**.



4. You can deploy any type of ASP.NET web app to Azure. For this quickstart, choose the **MVC** template.

5. Select **Docker support**, and make sure authentication is set to **No Authentication**. Select **Create**.

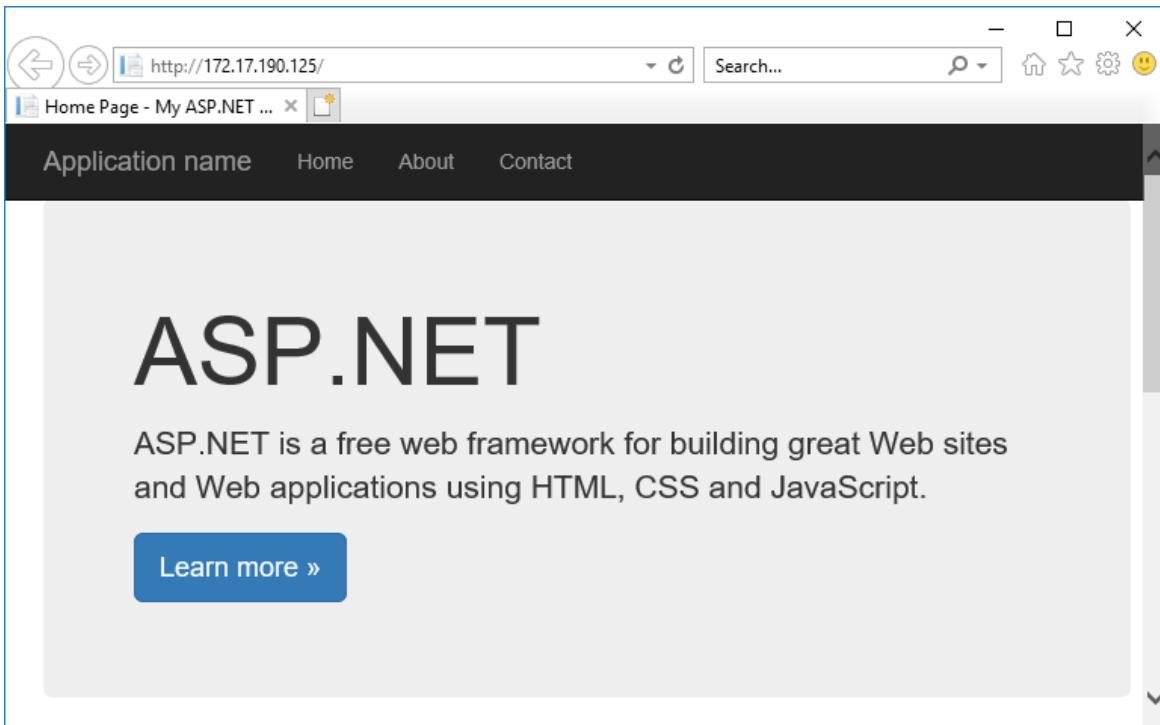


6. If the *Dockerfile* file isn't opened automatically, open it from the **Solution Explorer**.

7. You need a **supported parent image**. Change the parent image by replacing the `FROM` line with the following code and save the file:

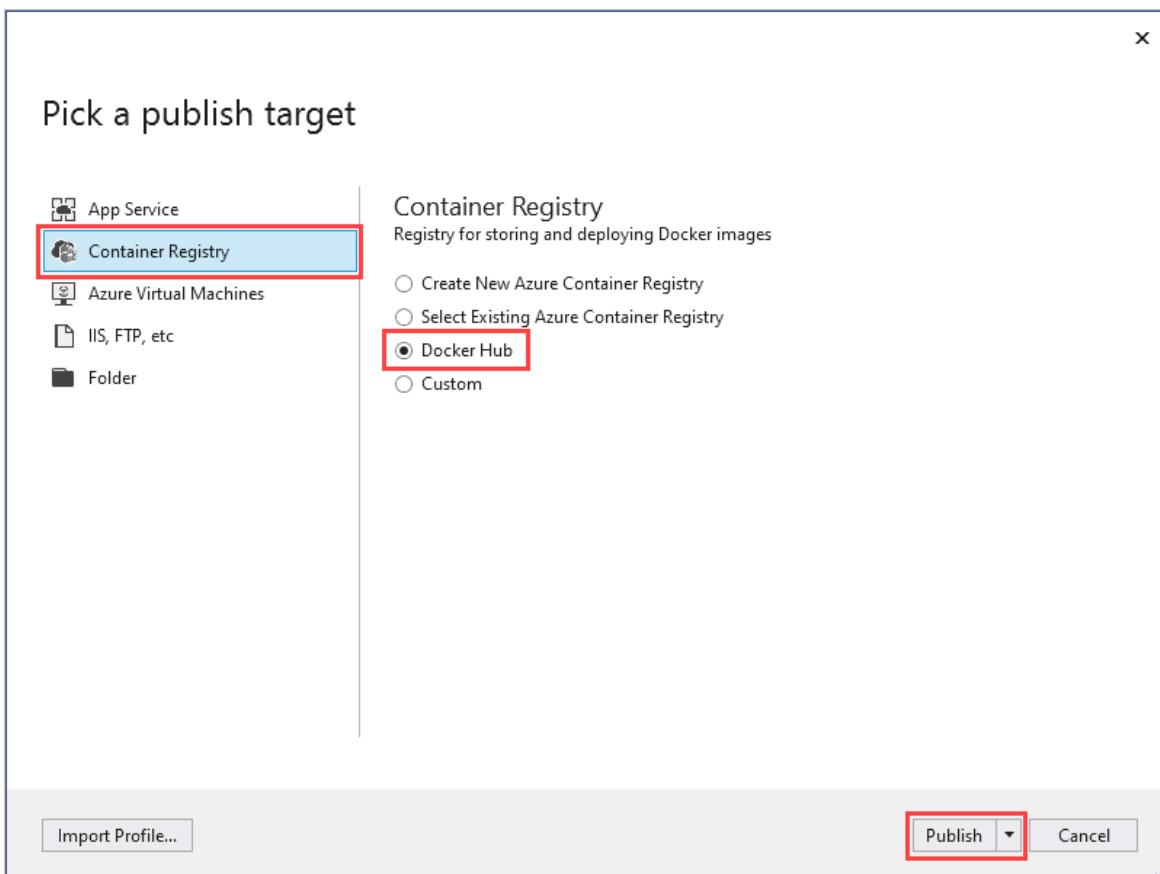
```
FROM mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-ltsc2019
```

8. From the Visual Studio menu, select **Debug > Start Without Debugging** to run the web app locally.



Publish to Docker Hub

1. In Solution Explorer, right-click the **myfirstazurewebapp** project and select **Publish**.
2. Choose **App Service** and then select **Publish**.
3. In **Pick a publish target**, select **Container Registry** and **Docker Hub**, and then click **Publish**.



4. Supply your Docker Hub account credentials and select **Save**.

Wait for the deployment to complete. The **Publish** page now shows the repository name to use later.

The screenshot shows the Azure portal interface for publishing a web application. The top navigation bar has 'myFirstAzureWebApp' and a close button. Below it, there are tabs for 'Overview', 'Connected Services', and 'Publish'. The 'Publish' tab is selected and highlighted in blue. Under the 'Publish' tab, there's a 'Connected Services' section with a 'New' button. The main area is titled 'Publish' with the sub-instruction 'Deploy your app to a folder, IIS, Azure, or another destination.' and a 'More info' link. A dropdown menu labeled 'registry.hub.docker.com_contoso' is open, showing options 'New', 'Edit', 'Rename', and 'Delete'. To the right of the dropdown is a 'Publish' button. Below this, there's a 'Summary' section with three items: 'Image Tag' (latest), 'Repository' (https://registry.hub.docker.com/contoso), and 'Configuration' (Release). On the right side, there's an 'Actions' section with a 'Edit Image Tag' link and a red box highlighting the 'https://registry.hub.docker.com/contoso' URL. A vertical scroll bar is visible on the right edge of the main content area.

5. Copy this repository name for later.

Create a Windows container app

1. Sign in to the [Azure portal](#).
2. Choose **Create a resource** in the upper left-hand corner of the Azure portal.
3. Under **Popular services**, select **Create** under **Web App**.
4. In **Create Web App**, choose your subscription and a **Resource Group**. You can create a new resource group if needed.
5. Provide an app name, such as *win-container-demo*. Choose **Docker Container** for **Publish** and **Windows** for **Operating System**. Select **Next: Docker** to continue.

Home > Create a resource >
Create Web App ...

Basics Docker Monitoring Tags Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ

Antares-Demo

Resource Group * ⓘ

(New) AppServiceRG

[Create new](#)

Instance Details

Name *

win-container-demo



.azurewebsites.net

Publish *

Code Docker Container

Operating System *

Linux Windows

Region *

Central US

[Not finding your App Service Plan? Try a different region.](#)

App Service Plan

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app.
[Learn more](#)

Windows Plan (Central US) * ⓘ

(New) ASP-AppServiceRG-b9f9



[Create new](#)

Sku and size *

Premium V3 P1V3

195 minimum ACU/vCPU, 8 GB memory

[Change size](#)

[Review + create](#)

< Previous

Next : Docker >

6. For **Image Source**, choose **Docker Hub** and for **Image and tag**, enter the repository name you copied in [Publish to Docker Hub](#).

Basics Docker Monitoring Tags Review + create

Pull container images from Azure Container Registry, Docker Hub or a private Docker repository. App Service will deploy the containerized app with your preferred dependencies to production in seconds.

Image Source

Docker Hub

Docker hub options

Access Type *

Public

Image and tag *

contoso/myFirstAzureWebApp:latest

Review + create

< Previous

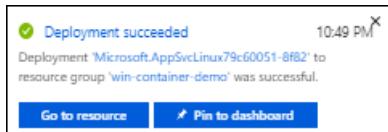
Next : Monitoring >

If you have a custom image elsewhere for your web application, such as in [Azure Container Registry](#) or in any other private repository, you can configure it here.

7. Select **Review and Create** and then **Create** and wait for Azure to create the required resources.

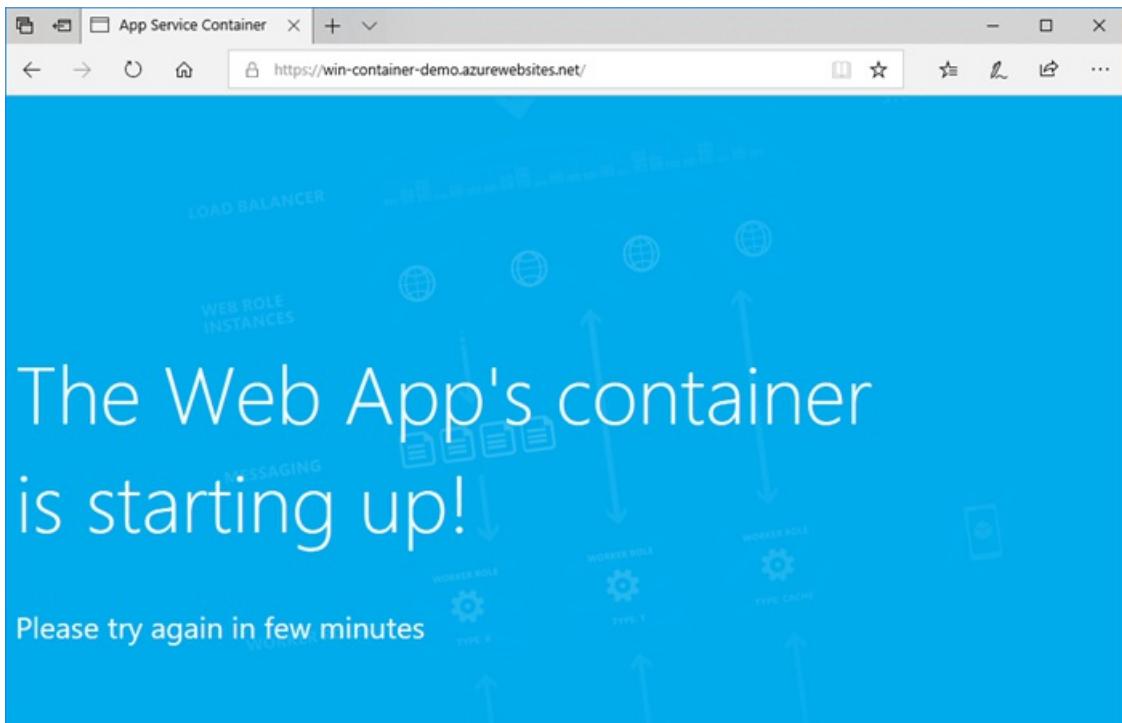
Browse to the container app

When the Azure operation is complete, a notification box is displayed.

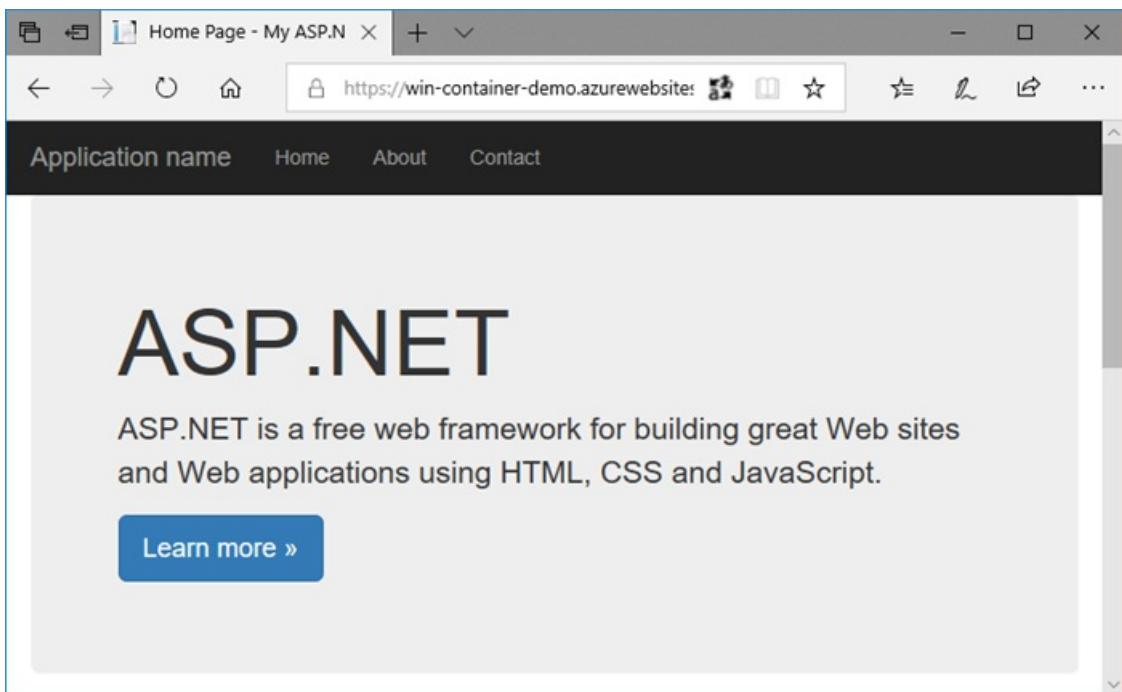


1. Click **Go to resource**.
2. In the overview of this resource, follow the link next to URL.

A new browser page opens to the following page:



Wait a few minutes and try again, until you get the default ASP.NET home page:



Congratulations! You're running your first custom Windows container in Azure App Service.

See container start-up logs

It may take some time for the Windows container to load. To see the progress, navigate to the following URL by replacing <app_name> with the name of your app.

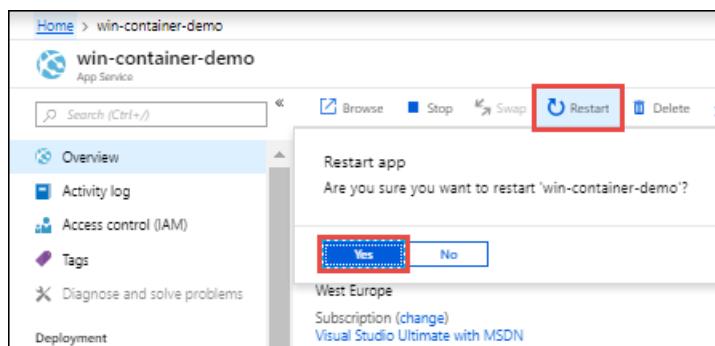
```
https://<app_name>.scm.azurewebsites.net/api/logstream
```

The streamed logs looks like this:

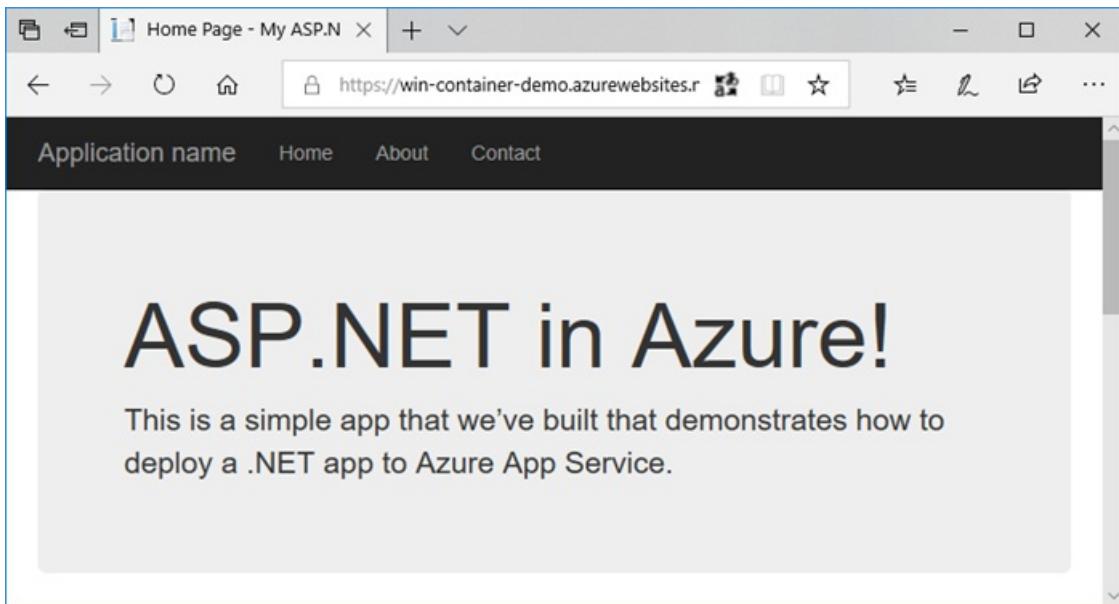
```
2018-07-27T12:03:11 Welcome, you are now connected to log-streaming service.  
27/07/2018 12:04:10.978 INFO - Site: win-container-demo - Start container succeeded. Container:  
facbf6cb214de86e58557a6d073396f640bbe2fdec88f8368695c8d1331fc94b  
27/07/2018 12:04:16.767 INFO - Site: win-container-demo - Container start complete  
27/07/2018 12:05:05.017 INFO - Site: win-container-demo - Container start complete  
27/07/2018 12:05:05.020 INFO - Site: win-container-demo - Container started successfully
```

Update locally and redeploy

1. In Visual Studio, in **Solution Explorer**, open **Views > Home > Index.cshtml**.
 2. Find the `<div class="jumbotron">` HTML tag near the top, and replace the entire element with the following code:
- ```
<div class="jumbotron">
 <h1>ASP.NET in Azure!</h1>
 <p class="lead">This is a simple app that we've built that demonstrates how to deploy a .NET app
 to Azure App Service.</p>
</div>
```
3. To redeploy to Azure, right-click the **myfirstazurewebapp** project in **Solution Explorer** and choose **Publish**.
  4. On the publish page, select **Publish** and wait for publishing to complete.
  5. To tell App Service to pull in the new image from Docker Hub, restart the app. Back in the app page in the portal, click **Restart > Yes**.



[Browse to the container app](#) again. As you refresh the webpage, the app should revert to the "Starting up" page at first, then display the updated webpage again after a few minutes.



## Next steps

[Migrate to Windows container in Azure](#)

Or, check out other resources:

[Configure custom container](#)

App Service on Linux provides pre-defined application stacks on Linux with support for languages such as .NET, PHP, Node.js and others. You can also use a custom Docker image to run your web app on an application stack that is not already defined in Azure. This quickstart shows you how to deploy an image from an [Azure Container Registry](#) (ACR) to App Service.

## Prerequisites

- An [Azure account](#)
- [Docker](#)
- [Visual Studio Code](#)
- The [Azure App Service extension for VS Code](#). You can use this extension to create, manage, and deploy Linux Web Apps on the Azure Platform as a Service (PaaS).
- The [Docker extension for VS Code](#). You can use this extension to simplify the management of local Docker images and commands and to deploy built app images to Azure.

## Create a container registry

This quickstart uses Azure Container Registry as the registry of choice. You're free to use other registries, but the steps may differ slightly.

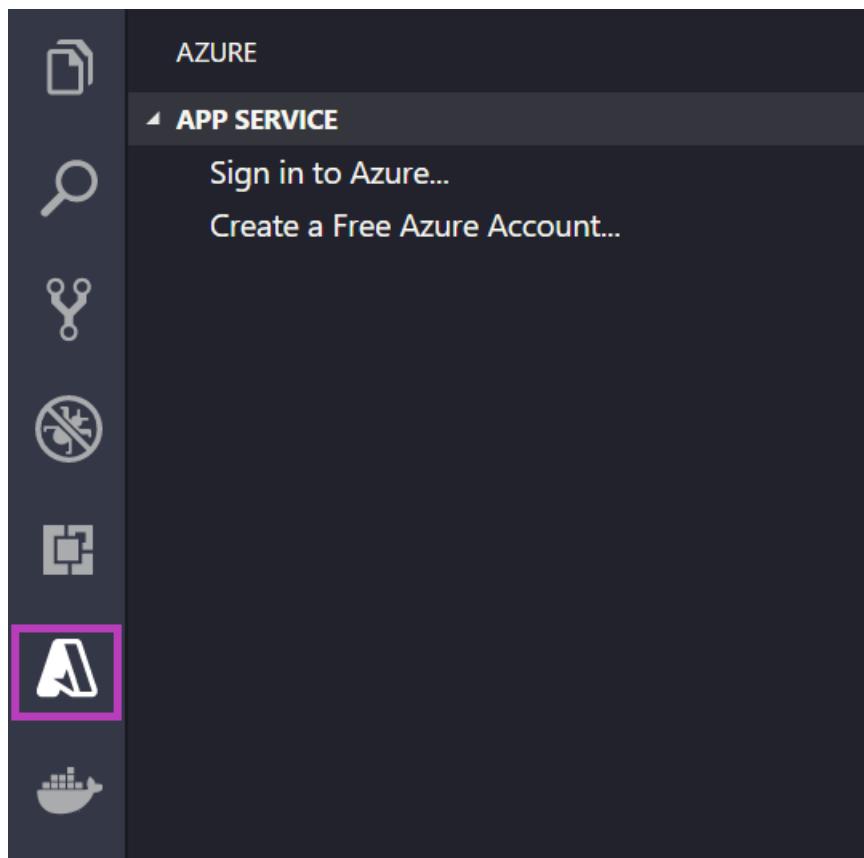
Create a container registry by following the instructions in [Quickstart: Create a private container registry using the Azure portal](#).

### IMPORTANT

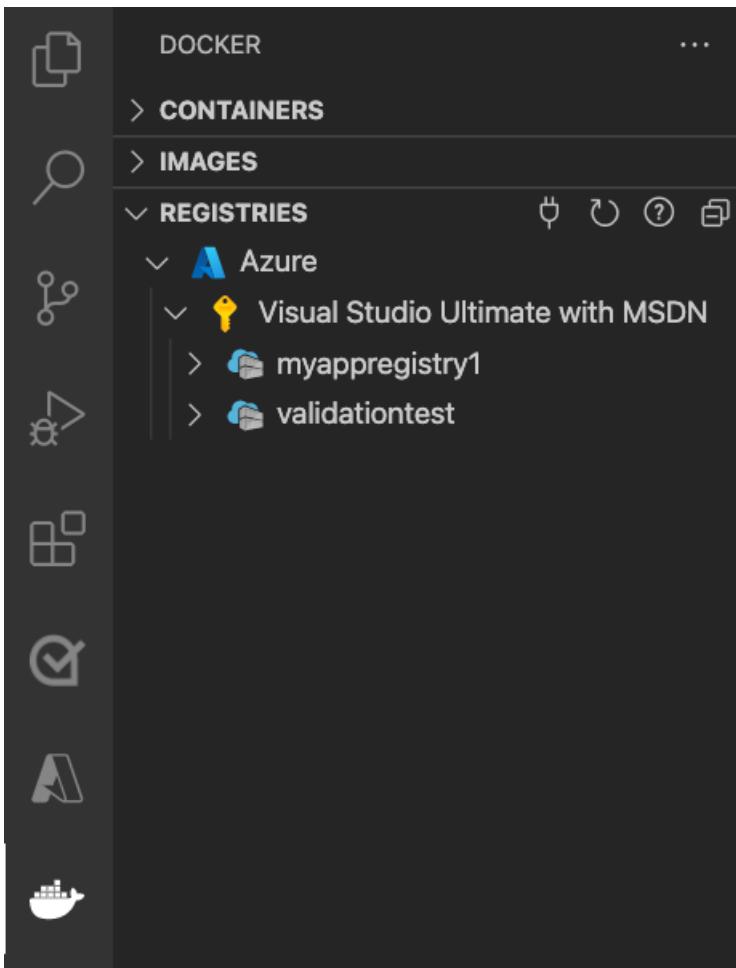
Be sure to set the **Admin User** option to **Enable** when you create the Azure container registry. You can also set it from the **Access keys** section of your registry page in the Azure portal. This setting is required for App Service access. For managed identity, see [Deploy from ACR tutorial](#).

## Sign in

1. Launch Visual Studio Code.
2. Select the Azure logo in the [Activity Bar](#), navigate to the APP SERVICE explorer, then select Sign in to Azure and follow the instructions.



3. In the [Status Bar](#) at the bottom, verify your Azure account email address. In the APP SERVICE explorer, your subscription should be displayed.
4. In the Activity Bar, select the Docker logo. In the REGISTRIES explorer, verify that the container registry you created appears.



## Check prerequisites

Verify that you have Docker installed and running. The following command will display the Docker version if it is running.

```
docker --version
```

## Create and build image

1. In Visual Studio Code, open an empty folder and add a file called `Dockerfile`. In the Dockerfile, paste in the content based on your desired language framework:

- [.NET](#)
- [Node.js](#)
- [Python](#)
- [Java](#)

```
FROM mcr.microsoft.com/appsvc/dotnetcore:lts

ENV PORT 8080
EXPOSE 8080

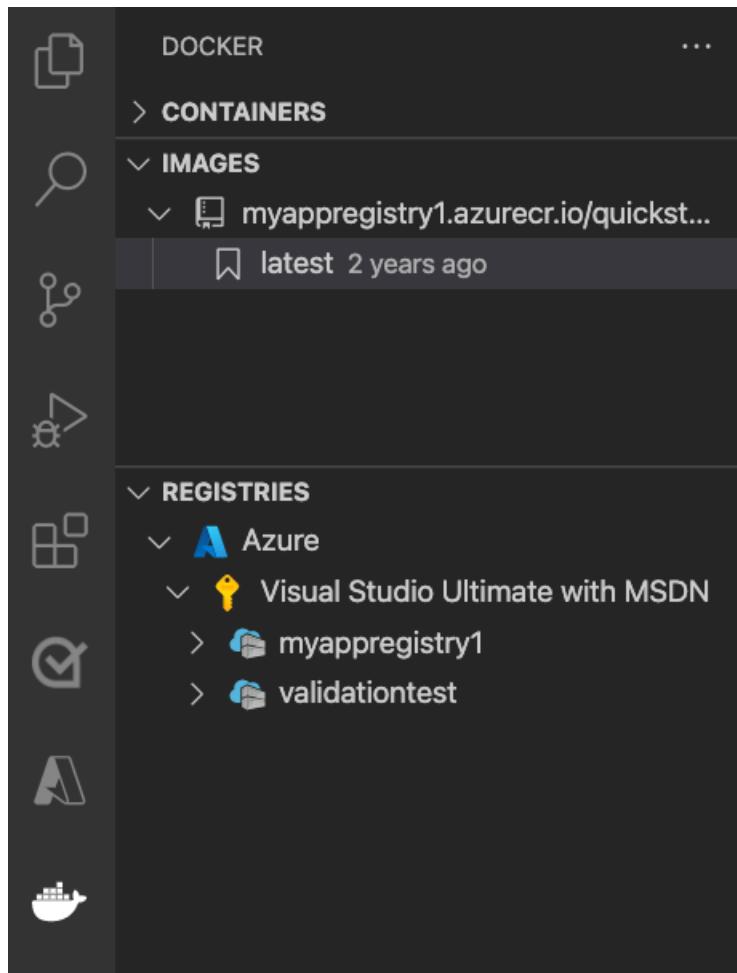
ENV ASPNETCORE_URLS "http://*:${PORT}"

ENTRYPOINT ["dotnet", "/defaulthome/hostingstart/hostingstart.dll"]
```

In this Dockerfile, the parent image is one of the built-in .NET containers of App Service. You can find the source

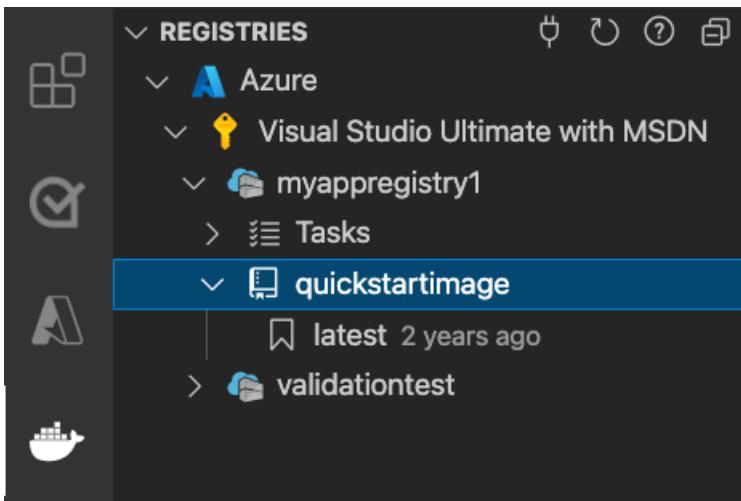
files for it in the Azure-App-Service/ImageBuilder GitHub repository, under GenerateDockerFiles/dotnetcore. Its Dockerfile copies a simple .NET app into `/default/home/hostingstart`. Your Dockerfile simply starts that app.

2. Open the Command Palette, and type Docker Images: Build Image. Type Enter to run the command.
3. In the image tag box, specify the tag you want in the following format:  
`<acr-name>.azurecr.io/<image-name>/<tag>`, where `<acr-name>` is the name of the container registry you created. Press Enter.
4. When the image finishes building, click Refresh at the top of the IMAGES explorer and verify that the image is built successfully.



## Deploy to container registry

1. In the Activity Bar, click the Docker icon. In the IMAGES explorer, find the image you just built.
2. Expand the image, right-click on the tag you want, and click Push.
3. Make sure the image tag begins with `<acr-name>.azurecr.io` and press Enter.
4. When Visual Studio Code finishes pushing the image to your container registry, click Refresh at the top of the REGISTRIES explorer and verify that the image is pushed successfully.



## Deploy to App Service

1. In the REGISTRIES explorer, expand the image, right-click the tag, and click **Deploy image to Azure App Service**.
2. Follow the prompts to choose a subscription, a globally unique app name, a resource group, and an App Service plan. Choose **B1 Basic** for the pricing tier, and a region near you.

After deployment, your app is available at <http://<app-name>.azurewebsites.net>.

A **Resource Group** is a named collection of all your application's resources in Azure. For example, a Resource Group can contain a reference to a website, a database, and an Azure Function.

An **App Service Plan** defines the physical resources that will be used to host your website. This quickstart uses a **Basic** hosting plan on **Linux** infrastructure, which means the site will be hosted on a Linux machine alongside other websites. If you start with the **Basic** plan, you can use the Azure portal to scale up so that yours is the only site running on a machine. For pricing, see [App Service pricing](#).

## Browse the website

The **Output** panel shows the status of the deployment operations. When the operation completes, click **Open Site** in the pop-up notification to open the site in your browser.

[I ran into an issue](#)

## Next steps

Congratulations, you've successfully completed this quickstart.

The App Service app pulls from the container registry every time it starts. If you rebuild your image, you just need to push it to your container registry, and the app pulls in the updated image when it restarts. To tell your app to pull in the updated image immediately, restart it.

[Configure custom container](#)

[Custom container tutorial](#)

[Multi-container app tutorial](#)

Other Azure extensions:

- [Cosmos DB](#)
- [Azure Functions](#)
- [Azure CLI Tools](#)

- [Azure Resource Manager Tools](#)
- [Azure Tools](#) extension pack includes all the extensions above.

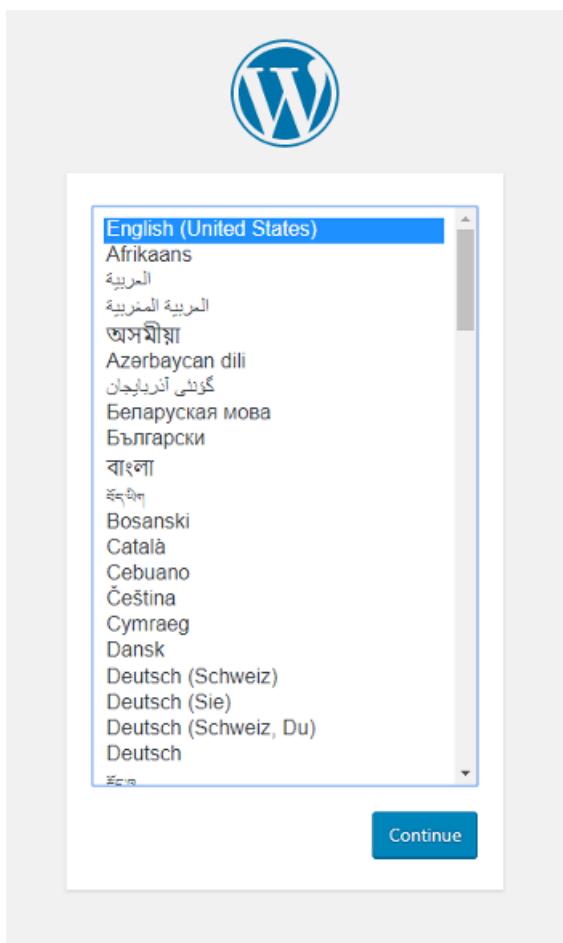
# Create a multi-container (preview) app using a Docker Compose configuration

11/2/2021 • 3 minutes to read • [Edit Online](#)

## NOTE

Multi-container is in preview.

[Web App for Containers](#) provides a flexible way to use Docker images. This quickstart shows how to deploy a multi-container app (preview) to Web App for Containers in the [Cloud Shell](#) using a Docker Compose configuration.



If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).

[!\[\]\(c687185540ff7df4122cde075a66c7cf\_img.jpg\) Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in

options, see [Sign in with the Azure CLI](#).

- When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

This article requires version 2.0.32 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Download the sample

For this quickstart, you use the compose file from [Docker](#). The configuration file can be found at [Azure Samples](#).

```
version: '3.3'

services:
 db:
 image: mysql:5.7
 volumes:
 - db_data:/var/lib/mysql
 restart: always
 environment:
 MYSQL_ROOT_PASSWORD: somewordpress
 MYSQL_DATABASE: wordpress
 MYSQL_USER: wordpress
 MYSQL_PASSWORD: wordpress

 wordpress:
 depends_on:
 - db
 image: wordpress:latest
 ports:
 - "8000:80"
 restart: always
 environment:
 WORDPRESS_DB_HOST: db:3306
 WORDPRESS_DB_USER: wordpress
 WORDPRESS_DB_PASSWORD: wordpress
 volumes:
 db_data:
```

In the Cloud Shell, create a quickstart directory and then change to it.

```
mkdir quickstart
cd $HOME/quickstart
```

Next, run the following command to clone the sample app repository to your quickstart directory. Then change to the `multicontainerwordpress` directory.

```
git clone https://github.com/Azure-Samples/multicontainerwordpress
cd multicontainerwordpress
```

## Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage

accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named *myResourceGroup* in the *South Central US* location. To see all supported locations for App Service on Linux in **Standard** tier, run the `az appservice list-locations --sku S1 --linux-workers-enabled` command.

```
az group create --name myResourceGroup --location "South Central US"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

## Create an Azure App Service plan

In the Cloud Shell, create an App Service plan in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Standard** pricing tier (`--sku S1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku S1 --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "adminSiteName": null,
 "appServicePlanName": "myAppServicePlan",
 "geoRegion": "South Central US",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "linux",
 "location": "South Central US",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

## Create a Docker Compose app

### NOTE

Docker Compose on Azure App Services currently has a limit of 4,000 characters at this time.

In your Cloud Shell terminal, create a multi-container [web app](#) in the `myAppServicePlan` App Service plan with the `az webapp create` command. Don't forget to replace `<app_name>` with a unique app name (valid characters are `a-z`, `0-9`, and `-`).

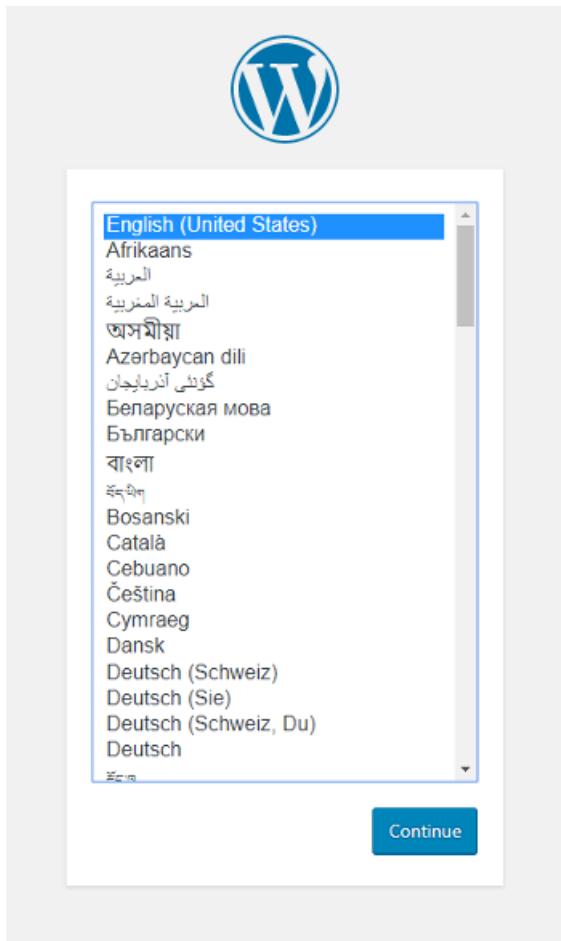
```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --multicontainer-config-type compose --multicontainer-config-file compose-wordpress.yml
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
{
 "additionalProperties": {},
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app_name>.azurewebsites.net",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

### Browse to the app

Browse to the deployed app at ([http://<app\\_name>.azurewebsites.net](http://<app_name>.azurewebsites.net)). The app may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser.



Congratulations, you've created a multi-container app in Web App for Containers.

### Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

```
az group delete --name myResourceGroup
```

## Next steps

[Tutorial: Multi-container WordPress app](#)

[Configure a custom container](#)

# Create an App Service app on Azure Arc (Preview)

11/2/2021 • 2 minutes to read • [Edit Online](#)

In this quickstart, you create an [App Service app to an Azure Arc-enabled Kubernetes cluster](#) (Preview). This scenario supports Linux apps only, and you can use a built-in language stack or a custom container.

## Prerequisites

- Set up your Azure Arc-enabled Kubernetes to run App Service.

## Add Azure CLI extensions

Launch the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

Because these CLI commands are not yet part of the core CLI set, add them with the following commands:

```
az extension add --upgrade --yes --name customlocation
az extension remove --name appservice-kube
az extension add --upgrade --yes --name appservice-kube
```

## 1. Create a resource group

Run the following command.

```
az group create --name myResourceGroup --location eastus
```

## 2. Get the custom location

Get the following information about the custom location from your cluster administrator (see [Create a custom location](#)).

```
customLocationGroup=<resource-group-containing-custom-location>
customLocationName=<name-of-custom-location>
```

Get the custom location ID for the next step.

```
customLocationId=$(az customlocation show \
--resource-group $customLocationGroup \
--name $customLocationName \
--query id \
--output tsv)
```

## 3. Create an app

The following example creates a Node.js app. Replace `<app-name>` with a name that's unique within your cluster (valid characters are `a-z`, `0-9`, and `-`). To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
az webapp create \
--resource-group myResourceGroup \
--name <app-name> \
--custom-location $customLocationId \
--runtime 'NODE|12-lts'
```

## 4. Deploy some code

### NOTE

`az webapp up` is not supported during the public preview.

Get a sample Nodejs app using Git and deploy it using [ZIP deploy](#). Replace `<app-name>` with your web app name.

```
git clone https://github.com/Azure-Samples/nodejs-docs-hello-world
cd nodejs-docs-hello-world
zip -r package.zip .
az webapp deployment source config-zip --resource-group myResourceGroup --name <app-name> --src package.zip
```

## 5. Get diagnostic logs using Log Analytics

### NOTE

To use Log Analytics, you should've previously enabled it when [installing the App Service extension](#). If you installed the extension without Log Analytics, skip this step.

Navigate to the [Log Analytics workspace that's configured with your App Service extension](#), then click Logs in the left navigation. Run the following sample query to show logs over the past 72 hours. Replace `<app-name>` with your web app name. If there's an error when running a query, try again in 10-15 minutes (there may be a delay for Log Analytics to start receiving logs from your application).

```
let StartTime = ago(72h);
let EndTime = now();
AppServiceConsoleLogs_CL
| where TimeGenerated between (StartTime .. EndTime)
| whereAppName_s =~ "<app-name>"
```

The application logs for all the apps hosted in your Kubernetes cluster are logged to the Log Analytics workspace in the custom log table named `AppServiceConsoleLogs_CL`.

`Log_s` contains application logs for a given App Service and `AppName_s` contains the App Service app name. In addition to logs you write via your application code, the `Log_s` column also contains logs on container startup, shutdown, and Function Apps.

You can learn more about log queries in [getting started with Kusto](#).

## (Optional) Deploy a custom container

To create a custom container app, run `az webapp create` with `--deployment-container-image-name`. For a private repository, add `--docker-registry-server-user` and `--docker-registry-server-password`.

For example, try:

```
az webapp create \
--resource-group myResourceGroup \
--name <app-name> \
--custom-location $customLocationId \
--deployment-container-image-name mcr.microsoft.com/appsvc/node:12-lts
```

To update the image after the app is created, see [Change the Docker image of a custom container](#)

## Next steps

- [Configure an ASP.NET Core app](#)
- [Configure a Node.js app](#)
- [Configure a PHP app](#)
- [Configure a Linux Python app](#)
- [Configure a Java app](#)
- [Configure a Linux Ruby app](#)
- [Configure a custom container](#)

# Tutorial: Map an existing custom DNS name to Azure App Service

11/2/2021 • 9 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows you how to map an existing custom Domain Name System (DNS) name to App Service.

In this tutorial, you learn how to:

- Map a subdomain by using a [CNAME record](#).
- Map a root domain by using an [A record](#).
- Map a [wildcard domain](#) by using a CNAME record.
- Redirect the default URL to a custom directory.

## 1. Prepare your environment

- [Create an App Service app](#), or use an app that you created for another tutorial.
- Make sure you can edit DNS records for your custom domain. If you don't have a custom domain yet, you can [purchase an App Service domain](#).

### NOTE

To edit DNS records, you need access to the DNS registry for your domain provider, such as GoDaddy. For example, to add DNS entries for `contoso.com` and `www.contoso.com`, you must be able to configure the DNS settings for the `contoso.com` root domain.

## 2. Prepare the app

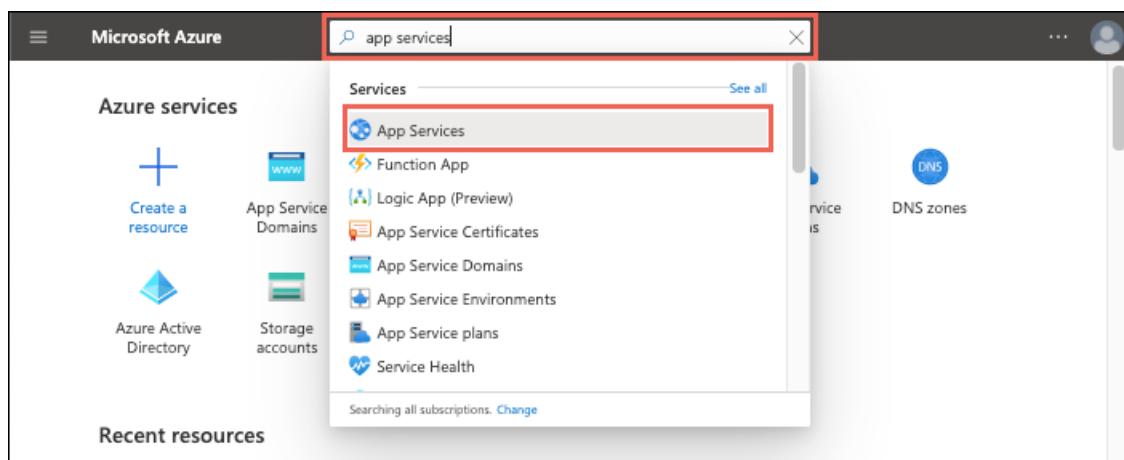
To map a custom DNS name to a web app, the web app's [App Service plan](#) must be a paid tier (not **Free (F1)**).

### Sign in to Azure

Open the [Azure portal](#), and sign in with your Azure account.

### Select the app in the Azure portal

1. Search for and select App Services.



2. On the App Services page, select the name of your Azure app.

The screenshot shows the Azure App Services management interface. At the top, there's a header with 'Home > App Services'. Below it, a search bar and filter options like 'Subscription == all', 'Resource group == all', and 'Location == all'. The main area displays a table with one record: 'my-demo-app'. The table columns are 'Name', 'Status', 'Location', 'Pricing Tier', and 'App Service Plan'. The 'my-demo-app' row is highlighted with a red border around its first column.

Name	Status	Location	Pricing Tier	App Service Plan
my-demo-app	Running	West Europe	Free	myAppServicePlan

You see the management page of the App Service app.

#### Check the pricing tier

1. In the left pane of the app page, scroll to the Settings section and select Scale up (App Service plan).

The screenshot shows the 'Settings' section of an Azure App Service page. On the left is a sidebar with icons for Deployment Center, Configuration, Container settings, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, Networking, and Scale up (App Service plan). The 'Scale up (App Service plan)' item is highlighted with a red border.

2. The app's current tier is highlighted by a blue border. Check to make sure that the app isn't in the F1 tier. Custom DNS isn't supported in the F1 tier.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

### Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:

#### Azure Compute Units (ACU)



Dedicated compute resources used to run applications deployed in the App...

#### Memory



Memory available to run applications

3. If the App Service plan isn't in the F1 tier, close the Scale up page and skip to [3. Get a domain verification ID](#).

#### Scale up the App Service plan

1. Select any of the non-free tiers (D1, B1, B2, B3, or any tier in the Production category). For additional options, select [See additional options](#).
2. Select **Apply**.



## Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included features

Every app hosted on this App Service plan will have access to these features:



#### Custom domains / SSL

Configure and purchase custom domains with SNI SSL bindings



#### Manual scale

Up to 3 instances. Subject to availability.

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:



#### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...



#### Memory

Memory per instance available to run applications deployed and running in...

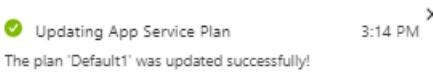


#### Storage

10 GB disk storage shared by all apps deployed in the App Service plan.

**Apply**

When you see the following notification, the scale operation is complete.



## 3. Get a domain verification ID

To add a custom domain to your app, you need to verify your ownership of the domain by adding a verification ID as a TXT record with your domain provider.

1. In the left pane of your app page, select **Custom domains**.
2. Copy the ID in the **Custom Domain Verification ID** box in the **Custom Domains** page for the next step.

[Home >](#)

## add-custom-domain | Custom domains

App Service

Search (Ctrl+ /) Refresh Troubleshoot FAQs

Deployment Center

Settings

- Configuration
- Authentication / Authorization
- Application Insights
- Identity
- Backups
- Custom domains**
- TLS/SSL settings

Custom Domains

Configure and manage custom domains assigned to your app [Learn more](#)

IP address: [REDACTED]

Custom Domain Verification ID: [REDACTED]

HTTPS Only:  Off  On

Add custom domain

**WARNING**

Adding domain verification IDs to your custom domain can prevent dangling DNS entries and help to avoid subdomain takeovers. For custom domains you previously configured without this verification ID, you should protect them from the same risk by adding the verification ID to your DNS record. For more information on this common high-severity threat, see [Subdomain takeover](#).

3. (A record only) To map an [A record](#), you need the app's external IP address. In the **Custom domains** page, copy the value of **IP address**.

## Custom Domains

Configure and manage custom domains assigned to your app [Learn more](#)

IP address: [REDACTED]

Custom Domain Verification ID: [REDACTED]

HTTPS Only:  Off  On

Add custom domain

Status Filter

All (1) Not Secure (0) Secure (1)

SSL STATE	ASSIGNED CUSTOM DOM...	SSL Binding
Secure	my-demo-app.azurewebsit...	



## 4. Create the DNS records

1. Sign in to the website of your domain provider.

**NOTE**

If you like, you can use Azure DNS to manage DNS records for your domain and configure a custom DNS name for Azure App Service. For more information, see [Tutorial: Host your domain in Azure DNS](#).

2. Find the page for managing DNS records.

## NOTE

Every domain provider has its own DNS records interface, so consult the provider's documentation. Look for areas of the site labeled **Domain Name, DNS, or Name Server Management**.

Often, you can find the DNS records page by viewing your account information and then looking for a link such as **My domains**. Go to that page, and then look for a link that's named something like **Zone file, DNS Records, or Advanced configuration**.

The following screenshot is an example of a DNS records page:

The screenshot shows a table titled "Records" with the following data:

Type	Name	Value	TTL
NS	@	ns31.domaincontrol.com	1 Hour
NS	@	ns32.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns31.domaincontrol.com.	600 seconds

At the bottom right of the table is a blue "ADD" button.

3. Select **Add** or the appropriate widget to create a record.
4. Select the type of record to create and follow the instructions. You can use either a [CNAME record](#) or an [A record](#) to map a custom DNS name to App Service.

## NOTE

### Which record to choose

- To map the root domain (for example, `contoso.com`), use an [A record](#). Don't use the CNAME record for the root record (for information, see [RFC 1912 Section 2.4](#)).
- To map a subdomain (for example, `www.contoso.com`), use a [CNAME record](#).
- You can map a subdomain to the app's IP address directly with an A record, but it's possible for [the IP address to change](#). The CNAME maps to the app's default hostname instead, which is less susceptible to change.
- To map a [wildcard domain](#) (for example, `*.contoso.com`), use a CNAME record.

- [CNAME](#)
- [A](#)
- [Wildcard \(CNAME\)](#)

For a subdomain like `www` in `www.contoso.com`, create two records according to the following table:

RECORD TYPE	HOST	VALUE	COMMENTS
CNAME	<code>&lt;subdomain&gt;</code> (for example, <code>www</code> )	<code>&lt;app-name&gt;.azurewebsites.net</code>	The domain mapping itself.
TXT	<code>asuid.&lt;subdomain&gt;</code> (for example, <code>asuid.www</code> )	The verification ID you got earlier	App Service accesses the <code>asuid.&lt;subdomain&gt;</code> TXT record to verify your ownership of the custom domain.

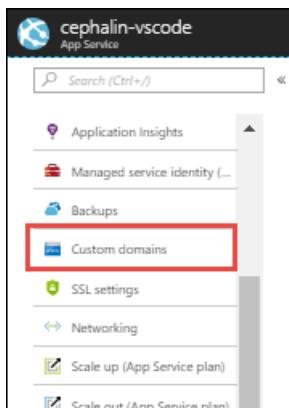
Name	Type	TTL	Value
@	NS	172800	ns1-05.azure-dns.com. ns2-05.azure-dns.net. ns3-05.azure-dns.org. ns4-05.azure-dns.info.
@	SOA	3600	Email: azuredns-hostmaster.microsoft.com Host: ns1-05.azure-dns.com. Refresh: 3600 Retry: 300 Expire: 2419200 Minimum TTL: 300 Serial number: 1
www	CNAME	3600	contoso.azurewebsites.net
asuid.www	TXT	3600	<domain-verification-id-from-your-app>

#### NOTE

For certain providers, such as GoDaddy, changes to DNS records don't become effective until you select a separate **Save Changes** link.

## 5. Enable the mapping in your app

1. In the left pane of the app page in the Azure portal, select **Custom domains**.



2. Select **Add custom domain**.

The screenshot shows the 'Custom Domains' configuration page. It includes fields for IP address, Custom Domain Verification ID, and an HTTPS Only toggle switch set to 'On'. At the bottom, there is a prominent red-bordered button labeled '+ Add custom domain'.

- CNAME
- A
- Wildcard (CNAME)

3. Type the fully qualified domain name that you added a CNAME record for, such as `www.contoso.com`.
4. Select **Validate**. The Add custom domain page appears.
5. Make sure that **Hostname record type** is set to **CNAME (www.example.com or any subdomain)**. Select **Add custom domain**.

**Add custom domain**

my-demo-app

Custom domain \*

www.contoso.com

Validate

Hostname record type

CNAME (www.example.com or any subdomain)

**CNAME configuration**

A CNAME record is used to specify that a domain name is an alias for another domain. In your scenario, that would be mapping www.contoso.com to custom domain verification id below. [Learn More](#)

Custom Domain Verification ID: ⓘ

CNAME

.azurewebsites.net

Add custom domain

It might take some time for the new custom domain to be reflected in the app's **Custom Domains** page. Refresh the browser to update the data.

**Custom Domains**

Configure and manage custom domains assigned to your app [Learn more](#)

IP address: ⓘ

Custom Domain Verification ID: ⓘ

HTTPS Only: ⓘ

On

+ Add custom domain

Status Filter

All (2)	Not Secure (1)	Secure (1)		
!	Not Secure	www.contoso.com	Add binding	***
✓	Secure	my-demo-app.azurewebsites.net		

#### NOTE

A warning label for your custom domain means that it's not yet bound to a TLS/SSL certificate. Any HTTPS request from a browser to your custom domain will receive an error or warning, depending on the browser. To add a TLS binding, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

If you missed a step or made a typo somewhere earlier, a verification error appears at the bottom of the page.

**i** DNS propagation

Please be aware that depending on your DNS provider it can take up to 48 hours for the DNS entry changes to propagate. You can verify that the DNS propagation is working as expected by using <http://digwebinterface.com/>. [Learn more](#)

---

**✓** Hostname availability

---

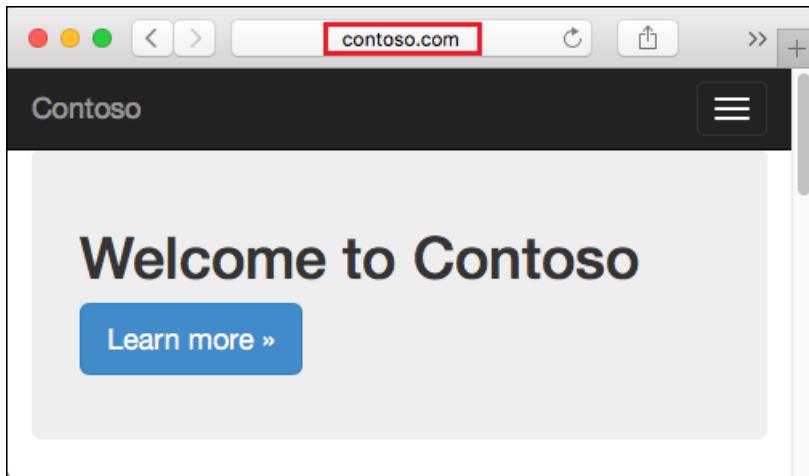
**!** Domain ownership

To verify domain ownership create CNAME and TXT record with your DNS provider using the configuration below, and replace {subdomain} with value of subdomain. [Learn more](#)

Type	Host	Value
TXT	asuid.www or asuid. {subdomain}	0123456789ABCDEF0123456789ABCDEF01;
CNAME	www or {subdomain}	my-demo-app.azurewebsites.net

## 6. Test in a browser

Browse to the DNS names that you configured earlier.



If you receive an HTTP 404 (Not Found) error when you browse to the URL of your custom domain, the two most common causes are:

- The custom domain configured is missing an A record or a CNAME record. You may have deleted the DNS record after you've enabled the mapping in your app. Check if the DNS records are properly configured using an [online DNS lookup](#) tool.
- The browser client has cached the old IP address of your domain. Clear the cache, and test DNS resolution again. On a Windows machine, you clear the cache with `ipconfig /flushdns`.

## Migrate an active domain

To migrate a live site and its DNS domain name to App Service with no downtime, see [Migrate an active DNS name to Azure App Service](#).

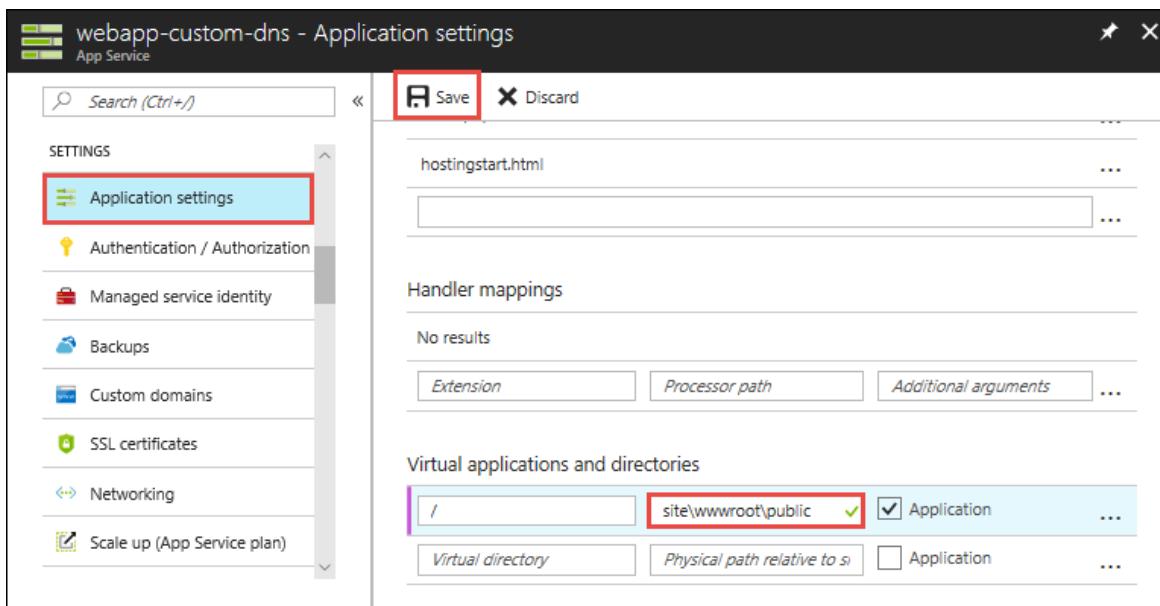
## Redirect to a custom directory

## NOTE

By default, App Service directs web requests to the root directory of your app code. But certain web frameworks don't start in the root directory. For example, [Laravel](#) starts in the `public` subdirectory. To continue the `contoso.com` DNS example, such an app is accessible at <http://contoso.com/public>, but you typically want to direct <http://contoso.com> to the `public` directory instead.

While this is a common scenario, it doesn't actually involve custom DNS mapping, but is about customizing the virtual directory within your app.

1. Select **Application settings** in the left pane of your web app page.
2. At the bottom of the page, the root virtual directory `/` points to `site\wwwroot` by default, which is the root directory of your app code. Change it to point to the `site\wwwroot\public` instead, for example, and save your changes.



3. After the operation finishes, verify by navigating to your app's root path in the browser (for example, <http://contoso.com> or <http://<app-name>.azurewebsites.net>).

## Automate with scripts

You can automate management of custom domains with scripts by using the [Azure CLI](#) or [Azure PowerShell](#).

### Azure CLI

The following command adds a configured custom DNS name to an App Service app.

```
az webapp config hostname add \
--webapp-name <app-name> \
--resource-group <resource_group_name> \
--hostname <fully_qualified_domain_name>
```

For more information, see [Map a custom domain to a web app](#).

### Azure PowerShell

#### **NOTE**

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

The following command adds a configured custom DNS name to an App Service app.

```
Set-AzWebApp `
 -Name <app-name> `
 -ResourceGroupName <resource_group_name> `
 -HostNames @("<fully_qualified_domain_name>","<app-name>.azurewebsites.net")
```

For more information, see [Assign a custom domain to a web app](#).

## Next steps

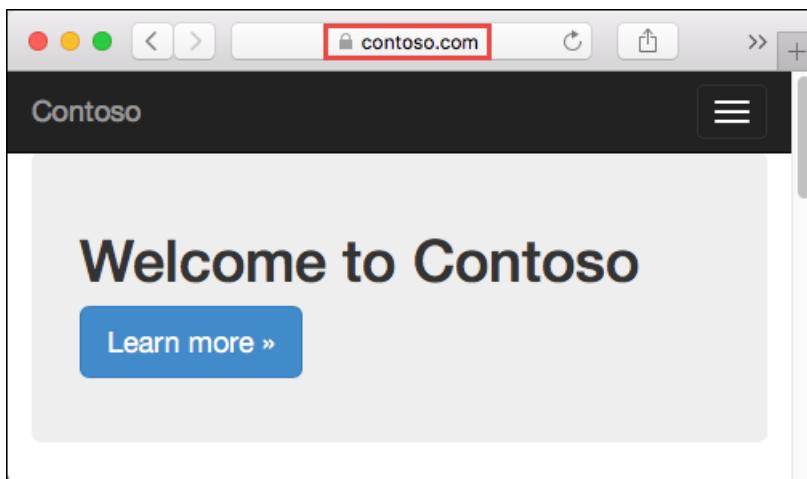
Continue to the next tutorial to learn how to bind a custom TLS/SSL certificate to a web app.

[Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)

# Secure a custom DNS name with a TLS/SSL binding in Azure App Service

11/2/2021 • 8 minutes to read • [Edit Online](#)

This article shows you how to secure the [custom domain](#) in your [App Service app](#) or [function app](#) by creating a certificate binding. When you're finished, you can access your App Service app at the `https://` endpoint for your custom DNS name (for example, `https://www.contoso.com`).



Securing a [custom domain](#) with a certificate involves two steps:

- [Add a private certificate to App Service](#) that satisfies all the [private certificate requirements](#).
- Create a TLS binding to the corresponding custom domain. This second step is covered by this article.

In this tutorial, you learn how to:

- Upgrade your app's pricing tier
- Secure a custom domain with a certificate
- Enforce HTTPS
- Enforce TLS 1.1/1.2
- Automate TLS management with scripts

## Prerequisites

To follow this how-to guide:

- [Create an App Service app](#)
- [Map a domain name to your app or buy and configure it in Azure](#)
- [Add a private certificate to your app](#)

### NOTE

The easiest way to add a private certificate is to [create a free App Service managed certificate](#).

## Prepare your web app

To create custom TLS/SSL bindings or enable client certificates for your App Service app, your [App Service plan](#)

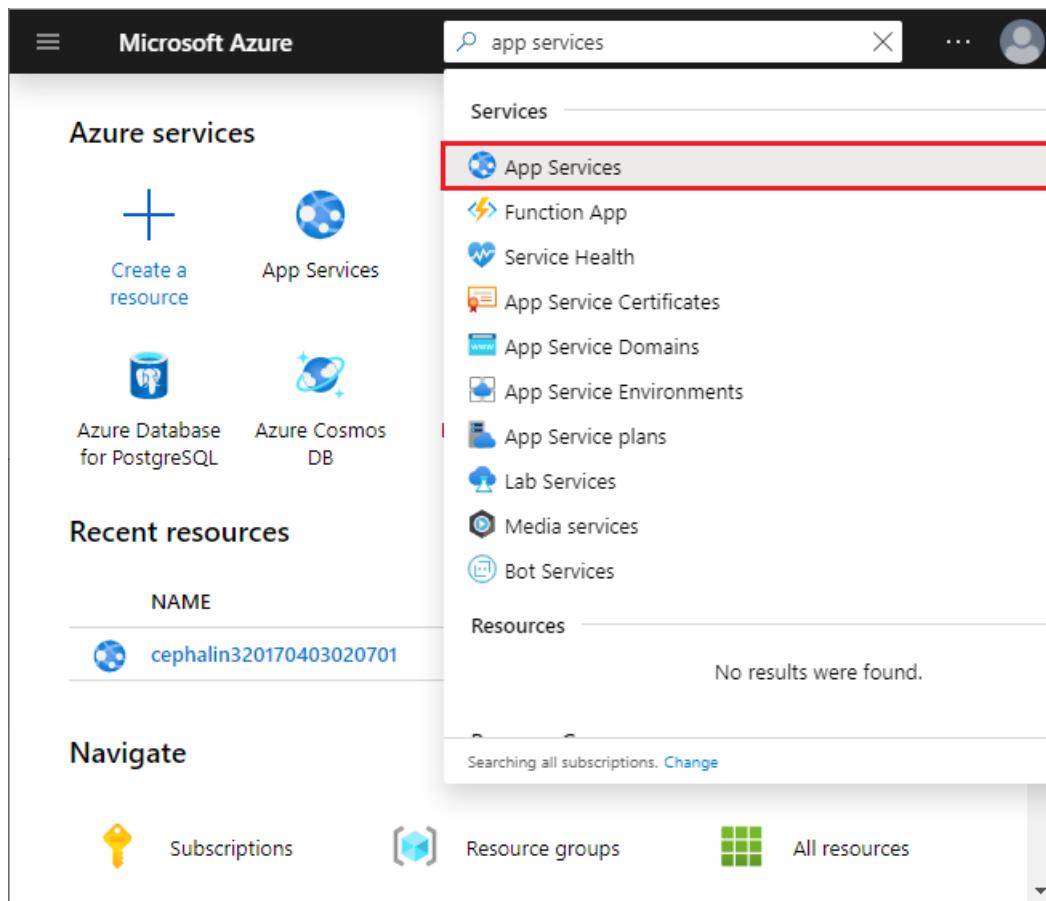
must be in the **Basic, Standard, Premium, or Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

## Sign in to Azure

Open the [Azure portal](#).

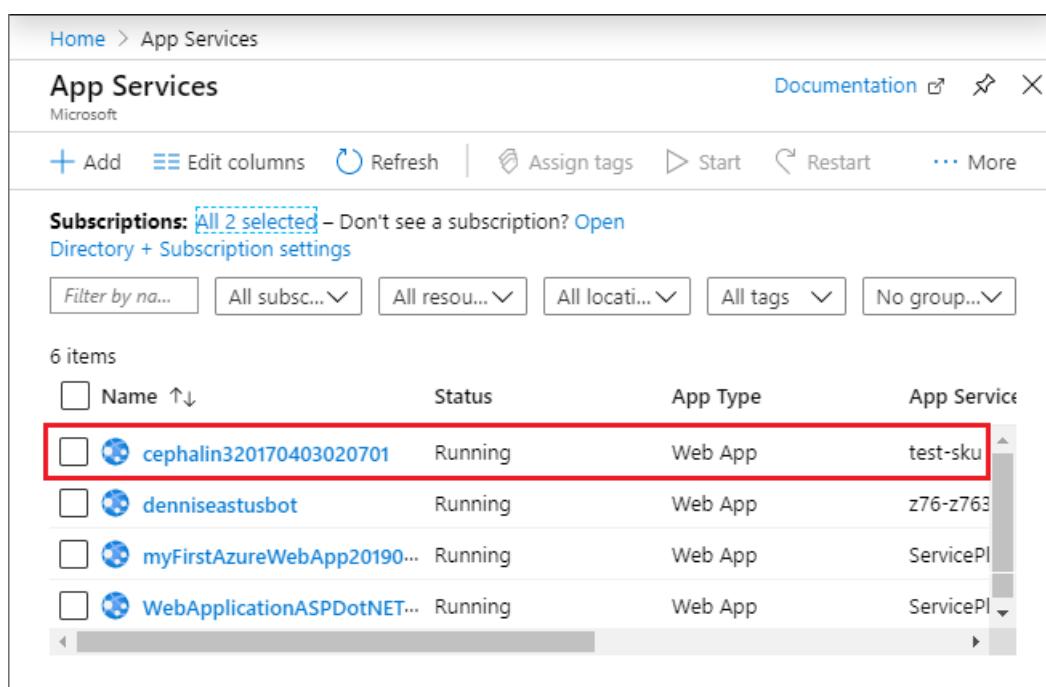
## Navigate to your web app

Search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "app services". Below the search bar, the "Services" section is expanded, showing various Azure services. The "App Services" option is highlighted with a red box. Other listed services include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. To the left, there's a sidebar titled "Azure services" with options like "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". Below the sidebar is a "Recent resources" section with a single item: "cephalin320170403020701". At the bottom, there are navigation links for "Subscriptions", "Resource groups", and "All resources".

On the App Services page, select the name of your web app.



The screenshot shows the "App Services" management page. The title bar says "Home > App Services". The main area is titled "App Services" with a Microsoft logo. It includes buttons for "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", and "More". Below this is a section for "Subscriptions" with a note about selecting a subscription. There are filter buttons for "Filter by name", "All subscr...", "All resou...", "All locati...", "All tags", and "No group...". The main table lists "6 items":

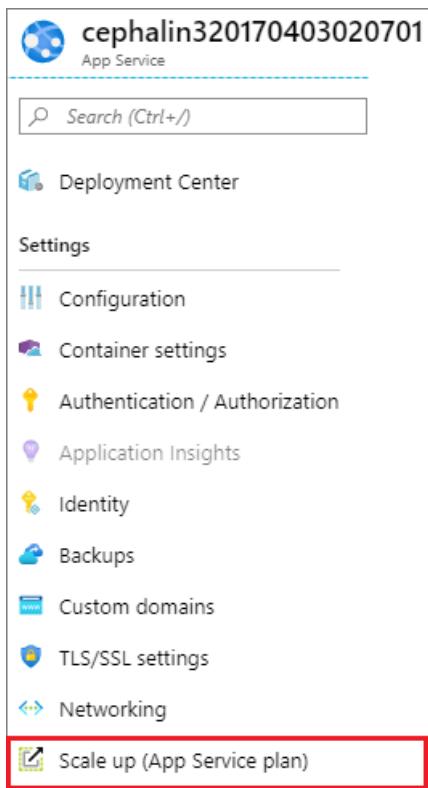
Name	Status	App Type	App Service
cephalin320170403020701	Running	Web App	test-sku
denniseastusbot	Running	Web App	z76-z763
myFirstAzureWebApp20190...	Running	Web App	ServicePl
WebApplicationASPDotNET...	Running	Web App	ServicePl

The first row, "cephalin320170403020701", has its entire row highlighted with a red box.

You have landed on the management page of your web app.

## Check the pricing tier

In the left-hand navigation of your web app page, scroll to the **Settings** section and select **Scale up (App Service plan)**.



Check to make sure that your web app is not in the F1 or D1 tier. Your web app's current tier is highlighted by a dark blue box.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

### Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:

#### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...

#### Memory

Memory available to run applications

Custom SSL is not supported in the F1 or D1 tier. If you need to scale up, follow the steps in the next section. Otherwise, close the Scale up page and skip the [Scale up your App Service plan](#) section.

#### Scale up your App Service plan

Select any of the non-free tiers (B1, B2, B3, or any tier in the Production category). For additional options, click See additional options.

Click **Apply**.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

### Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included features

Every app hosted on this App Service plan will have access to these features:



#### Custom domains / SSL

Configure and purchase custom domains with SNI SSL bindings



#### Manual scale

Up to 3 instances. Subject to availability.

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:



#### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...



#### Memory

Memory per instance available to run applications deployed and running in...



#### Storage

10 GB disk storage shared by all apps deployed in the App Service plan.

**Apply**

When you see the following notification, the scale operation is complete.



### Secure a custom domain

Do the following steps:

In the [Azure portal](#), from the left menu, select App Services > <app-name>.

From the left navigation of your app, start the TLS/SSL Binding dialog by:

- Selecting Custom domains > Add binding
- Selecting TLS/SSL settings > Add TLS/SSL binding

SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
<span style="color: red;">!</span> Not Secure	contoso.com	<a href="#">Add binding</a>
<span style="color: green;">✓</span> Secure	my-demo-app.azurewebsites.net	

In **Custom Domain**, select the custom domain you want to add a binding for.

If your app already has a certificate for the selected custom domain, go to [Create binding](#) directly. Otherwise, keep going.

### Add a certificate for custom domain

If your app has no certificate for the selected custom domain, then you have two options:

- **Upload PFX Certificate** - Follow the workflow at [Upload a private certificate](#), then select this option here.
- **Import App Service Certificate** - Follow the workflow at [Import an App Service certificate](#), then select this option here.

#### NOTE

You can also [Create a free certificate](#) or [Import a Key Vault certificate](#), but you must do it separately and then return to the TLS/SSL Binding dialog.

### Create binding

Use the following table to help you configure the TLS binding in the **TLS/SSL Binding** dialog, then click **Add Binding**.

SETTING	DESCRIPTION
Custom domain	The domain name to add the TLS/SSL binding for.
Private Certificate Thumbprint	The certificate to bind.
TLS/SSL Type	<ul style="list-style-type: none"> <li><b>SNI SSL</b> - Multiple SNI SSL bindings may be added. This option allows multiple TLS/SSL certificates to secure multiple domains on the same IP address. Most modern browsers (including Internet Explorer, Chrome, Firefox, and Opera) support SNI (for more information, see <a href="#">Server Name Indication</a>).</li> <li><b>IP SSL</b> - Only one IP SSL binding may be added. This option allows only one TLS/SSL certificate to secure a dedicated public IP address. After you configure the binding, follow the steps in <a href="#">Remap records for IP SSL</a>. IP SSL is supported only in <b>Standard</b> tier or above.</li> </ul>

Once the operation is complete, the custom domain's TLS/SSL state is changed to **Secure**.

Status Filter		
SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
Secure	contoso.com	SNI SSL
Secure	my-demo-app.azurewebsites.net	...

#### NOTE

A **Secure** state in the **Custom domains** means that it is secured with a certificate, but App Service doesn't check if the certificate is self-signed or expired, for example, which can also cause browsers to show an error or warning.

## Remap records for IP SSL

If you don't use IP SSL in your app, skip to [Test HTTPS for your custom domain](#).

There are two changes you need to make, potentially:

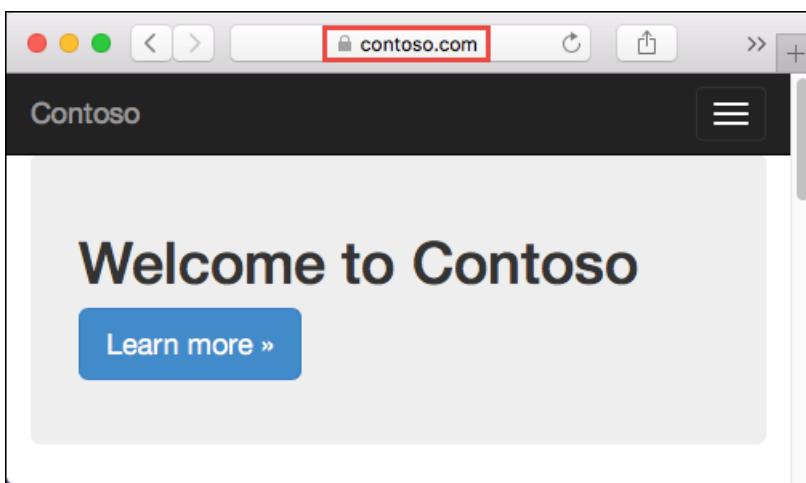
- By default, your app uses a shared public IP address. When you bind a certificate with IP SSL, App Service creates a new, dedicated IP address for your app. If you mapped an A record to your app, update your domain registry with this new, dedicated IP address.

Your app's **Custom domain** page is updated with the new, dedicated IP address. [Copy this IP address](#), then [remap the A record](#) to this new IP address.

- If you have an SNI SSL binding to `<app-name>.azurewebsites.net`, [remap any CNAME mapping](#) to point to `sni.<app-name>.azurewebsites.net` instead (add the `sni` prefix).

## Test HTTPS

In various browsers, browse to `https://<your.custom.domain>` to verify that it serves up your app.



Your application code can inspect the protocol via the "x-appservice-proto" header. The header will have a value of `http` or `https`.

#### NOTE

If your app gives you certificate validation errors, you're probably using a self-signed certificate.

If that's not the case, you may have left out intermediate certificates when you export your certificate to the PFX file.

## Prevent IP changes

Your inbound IP address can change when you delete a binding, even if that binding is IP SSL. This is especially important when you renew a certificate that's already in an IP SSL binding. To avoid a change in your app's IP address, follow these steps in order:

1. Upload the new certificate.
2. Bind the new certificate to the custom domain you want without deleting the old one. This action replaces the binding instead of removing the old one.
3. Delete the old certificate.

## Enforce HTTPS

By default, anyone can still access your app using HTTP. You can redirect all HTTP requests to the HTTPS port.

In your app page, in the left navigation, select **TLS/SSL settings**. Then, in **HTTPS Only**, select **On**.

The screenshot shows the 'TLS/SSL settings' blade in the Azure portal. The left sidebar has 'TLS/SSL settings' selected. The main area shows 'Protocol Settings' with 'HTTPS Only' set to 'On'. Below it is a section for 'TLS/SSL bindings' which is currently empty.

When the operation is complete, navigate to any of the HTTP URLs that point to your app. For example:

- `http://<app_name>.azurewebsites.net`
- `http://contoso.com`
- `http://www.contoso.com`

## Enforce TLS versions

Your app allows **TLS** 1.2 by default, which is the recommended TLS level by industry standards, such as [PCI DSS](#). To enforce different TLS versions, follow these steps:

In your app page, in the left navigation, select **TLS/SSL settings**. Then, in **TLS version**, select the minimum TLS version you want. This setting controls the inbound calls only.

The screenshot shows the 'TLS/SSL settings' page in the Azure portal. The left sidebar has a 'TLS/SSL settings' item selected, indicated by a red box. The main area has a 'Bindings' tab selected, also indicated by a red box. Under 'Protocol Settings', 'HTTPS Only' is set to 'On'. The 'Minimum TLS Version' is set to '1.2', which is highlighted with a red box. The 'TLS/SSL bindings' section shows a table with columns for Host name, Private Certificate Thumbprint, and TLS/SSL Type. A note at the bottom says 'No TLS/SSL bindings configured for the app.'

When the operation is complete, your app rejects all connections with lower TLS versions.

## Handle TLS termination

In App Service, [TLS termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `x-Forwarded-Proto` header.

Language specific configuration guides, such as the [Linux Node.js configuration](#) guide, shows you how to detect an HTTPS session in your application code.

## Automate with scripts

### Azure CLI

```

#!/bin/bash

fqdn=<replace-with-www.{yourdomain}>
pfxPath=<replace-with-path-to-your-.PFX-file>
pfxPassword=<replace-with-your=.PFX-password>
resourceGroup=myResourceGroup
webappName=mywebapp$RANDOM

Create a resource group.
az group create --location westeurope --name $resourceGroup

Create an App Service plan in Basic tier (minimum required by custom domains).
az appservice plan create --name $webappName --resource-group $resourceGroup --sku B1

Create a web app.
az webapp create --name $webappName --resource-group $resourceGroup \
--plan $webappName

echo "Configure a CNAME record that maps $fqdn to $webappName.azurewebsites.net"
read -p "Press [Enter] key when ready ..."

Before continuing, go to your DNS configuration UI for your custom domain and follow the
instructions at https://aka.ms/appservicecustomdns to configure a CNAME record for the
hostname "www" and point it to your web app's default domain name.

Map your prepared custom domain name to the web app.
az webapp config hostname add --webapp-name $webappName --resource-group $resourceGroup \
--hostname $fqdn

Upload the SSL certificate and get the thumbprint.
thumbprint=$(az webapp config ssl upload --certificate-file $pfxPath \
--certificate-password $pfxPassword --name $webappName --resource-group $resourceGroup \
--query thumbprint --output tsv)

Binds the uploaded SSL certificate to the web app.
az webapp config ssl bind --certificate-thumbprint $thumbprint --ssl-type SNI \
--name $webappName --resource-group $resourceGroup

echo "You can now browse to https://$fqdn"

```

## PowerShell

```

$fqdn=<Replace with your custom domain name>
$pfxPath=<Replace with path to your .PFX file>
$pfxPassword=<Replace with your .PFX password>
$webappname="mywebapp$(Get-Random)"
$location="West Europe"

Create a resource group.
New-AzResourceGroup -Name $webappname -Location $location

Create an App Service plan in Free tier.
New-AzAppServicePlan -Name $webappname -Location $location `
-ResourceGroupName $webappname -Tier Free

Create a web app.
New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname `
-ResourceGroupName $webappname

Write-Host "Configure a CNAME record that maps $fqdn to $webappname.azurewebsites.net"
Read-Host "Press [Enter] key when ready ..."

Before continuing, go to your DNS configuration UI for your custom domain and follow the
instructions at https://aka.ms/appservicecustomdns to configure a CNAME record for the
hostname "www" and point it to your web app's default domain name.

Upgrade App Service plan to Basic tier (minimum required by custom SSL certificates)
Set-AzAppServicePlan -Name $webappname -ResourceGroupName $webappname `
-Tier Basic

Add a custom domain name to the web app.
Set-AzWebApp -Name $webappname -ResourceGroupName $webappname `
-HostNames @($fqdn,"$webappname.azurewebsites.net")

Upload and bind the SSL certificate to the web app.
New-AzWebAppSSLBinding -WebAppName $webappname -ResourceGroupName $webappname -Name $fqdn `
-CertificateFilePath $pfxPath -CertificatePassword $pfxPassword -SslState SniEnabled

```

## More resources

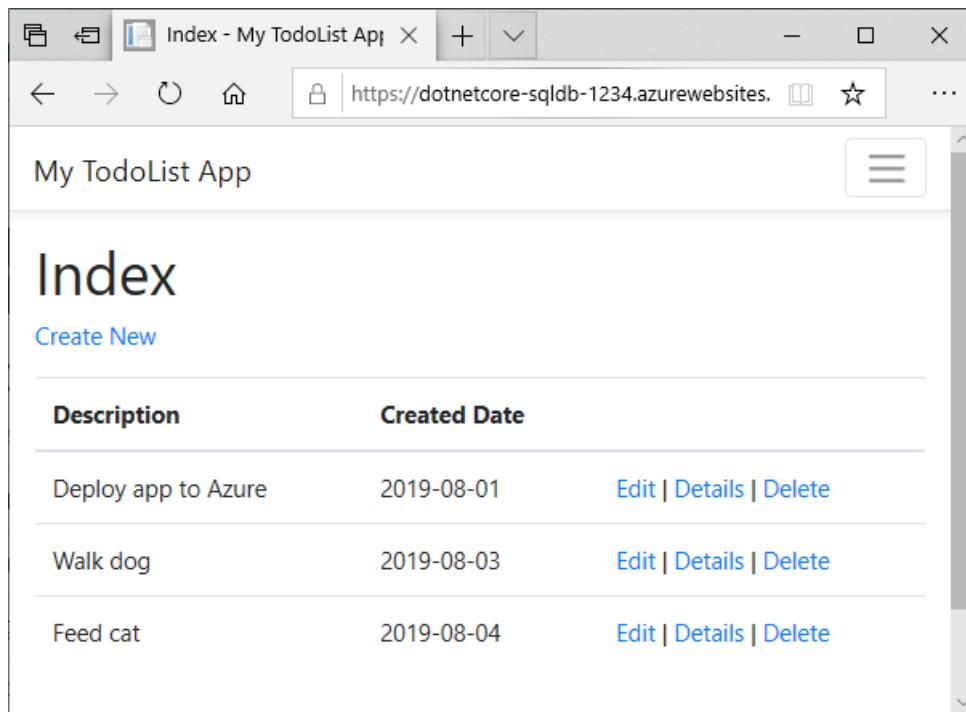
- [Use a TLS/SSL certificate in your code in Azure App Service](#)
- [FAQ : App Service Certificates](#)

# Tutorial: Build an ASP.NET Core and Azure SQL Database app in Azure App Service

11/2/2021 • 16 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service in Azure. This tutorial shows how to create an ASP.NET Core app and connect it to SQL Database. When you're done, you'll have a .NET MVC app running in App Service on Windows.

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create an ASP.NET Core app and connect it to a SQL Database. When you're done, you'll have an ASP.NET Core MVC app running in App Service on Linux.



The screenshot shows a web browser window with the title "Index - My TodoList App". The address bar contains the URL "https://dotnetcore-sqldb-1234.azurewebsites.net". The main content area displays the "Index" page of a TodoList application. At the top, there is a "Create New" link. Below it is a table with three columns: "Description", "Created Date", and actions "Edit | Details | Delete". The table contains three rows of data:

Description	Created Date	
Deploy app to Azure	2019-08-01	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Walk dog	2019-08-03	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Feed cat	2019-08-04	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

In this tutorial, you learn how to:

- Create a SQL Database in Azure
- Connect an ASP.NET Core app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install the latest .NET 5.0 SDK](#)
- Use the Bash environment in [Azure Cloud Shell](#).

[Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Create local ASP.NET Core app

In this step, you set up the local ASP.NET Core project.

### Clone the sample application

1. In the terminal window, `cd` to a working directory.
2. Run the following commands to clone the sample repository and change to its root.

```
git clone https://github.com/azure-samples/dotnetcore-sqldb-tutorial
cd dotnetcore-sqldb-tutorial
```

The sample project contains a basic CRUD (create-read-update-delete) app using [Entity Framework Core](#).

3. Make sure the default branch is `main`.

```
git branch -m main
```

#### TIP

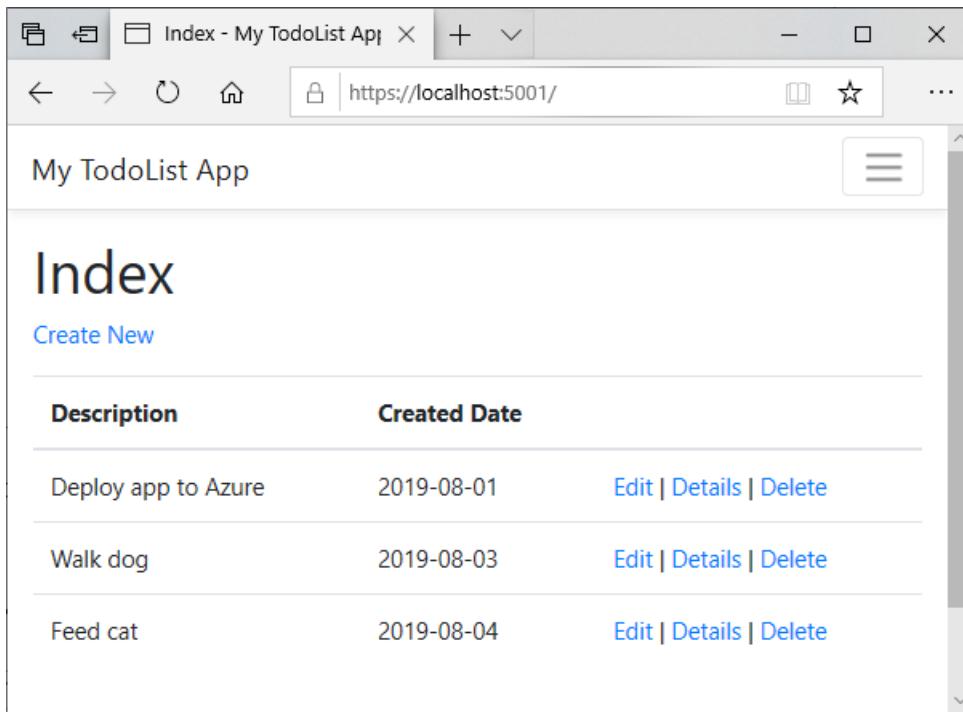
The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main` (see [Change deployment branch](#)), this tutorial also shows you how to deploy a repository from `main`.

### Run the application

1. Run the following commands to install the required packages, run database migrations, and start the application.

```
dotnet tool install -g dotnet-eF
dotnet ef database update
dotnet run
```

2. Navigate to `http://localhost:5000` in a browser. Select the **Create New** link and create a couple *to-do* items.



3. To stop ASP.NET Core at any time, press `ctrl+c` in the terminal.

## Create production SQL Database

In this step, you create a SQL Database in Azure. When your app is deployed to Azure, it uses this cloud database.

For SQL Database, this tutorial uses [Azure SQL Database](#).

### Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service in **Free** tier, run the `az appservice list-locations --sku FREE` command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

### Create a SQL Database logical server

In the Cloud Shell, create a SQL Database logical server with the `az sql server create` command.

Replace the *<server-name>* placeholder with a *unique* SQL Database name. This name is used as the part of the globally unique SQL Database endpoint, *<server-name>.database.windows.net*. Valid characters are `a - z`, `0 - 9`, `-`. Also, replace *<db-username>* and *<db-password>* with a username and password of your choice.

```
az sql server create --name <server-name> --resource-group myResourceGroup --location "West Europe" --admin-user <db-username> --admin-password <db-password>
```

When the SQL Database logical server is created, the Azure CLI shows information similar to the following

example:

```
{
 "administratorLogin": "<db-username>",
 "administratorLoginPassword": null,
 "fullyQualifiedDomainName": "<server-name>.database.windows.net",
 "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myResourceGroup/providers/Microsoft.Sql/servers/<server-name>",
 "identity": null,
 "kind": "v12.0",
 "location": "westeurope",
 "name": "<server-name>",
 "resourceGroup": "myResourceGroup",
 "state": "Ready",
 "tags": null,
 "type": "Microsoft.Sql/servers",
 "version": "12.0"
}
```

## Configure a server firewall rule

1. Create an [Azure SQL Database server-level firewall rule](#) using the `az sql server firewall create` command. When both starting IP and end IP are set to 0.0.0.0, the firewall is only opened for other Azure resources.

```
az sql server firewall-rule create --resource-group myResourceGroup --server <server-name> --name
AllowAzureIps --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0
```

### TIP

You can be even more restrictive in your firewall rule by [using only the outbound IP addresses your app uses](#).

2. In the Cloud Shell, run the command again to allow access from your local computer by replacing *<your-ip-address>* with [your local IPv4 IP address](#).

```
az sql server firewall-rule create --name AllowLocalClient --server <server-name> --resource-group
myResourceGroup --start-ip-address=<your-ip-address> --end-ip-address=<your-ip-address>
```

## Create a database

Create a database with an S0 performance level in the server using the `az sql db create` command.

```
az sql db create --resource-group myResourceGroup --server <server-name> --name coreDB --service-objective
S0
```

## Retrieve connection string

Get the connection string using the `az sql db show-connection-string` command.

```
az sql db show-connection-string --client ado.net --server <server-name> --name coreDB
```

In the command output, replace *<username>*, and *<password>* with the database administrator credentials you used earlier.

This is the connection string for your ASP.NET Core app. Copy it for use later.

## Configure app to connect to production database

In your local repository, open Startup.cs and find the following code:

```
services.AddDbContext<MyDbContext>(options =>
 options.UseSqlite("Data Source=localdatabase.db"));
```

Replace it with the following code.

```
services.AddDbContext<MyDbContext>(options =>
 options.UseSqlServer(Configuration.GetConnectionString("MyDbConnection")));
```

### IMPORTANT

For production apps that need to scale out, follow the best practices in [Applying migrations in production](#).

## Run database migrations to the production database

Your app currently connects to a local Sqlite database. Now that you configured an Azure SQL Database, recreate the initial migration to target it.

From the repository root, run the following commands. Replace *<connection-string>* with the connection string you created earlier.

```
Delete old migrations
rm -r Migrations
Recreate migrations with UseSqlServer (see previous snippet)
dotnet ef migrations add InitialCreate

Set connection string to production database
PowerShell
$env:ConnectionStrings:MyDbConnection=<connection-string>
CMD (no quotes)
set ConnectionStrings:MyDbConnection=<connection-string>
Bash (no quotes)
export ConnectionStrings__MyDbConnection=<connection-string>

Run migrations
dotnet ef database update
```

## Run app with new configuration

1. Now that database migrations is run on the production database, test your app by running:

```
dotnet run
```

2. Navigate to `http://localhost:5000` in a browser. Select the **Create New** link and create a couple *to-do* items. Your app is now reading and writing data to the production database.

3. Commit your local changes, then commit it into your Git repository.

```
git add .
git commit -m "connect to SQLDB in Azure"
```

You're now ready to deploy your code.

# Deploy app to Azure

In this step, you deploy your SQL Database-connected ASP.NET Core application to App Service.

## Configure local git deployment

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the [az webapp deployment user set](#) command in Azure Cloud Shell. Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

## Create an App Service plan

In the Cloud Shell, create an App Service plan with the [az appservice plan create](#) command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "adminSiteName": null,
 "appServicePlanName": "myAppServicePlan",
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "app",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

In the Cloud Shell, create an App Service plan with the [az appservice plan create](#) command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "freeOfferExpirationTime": null,
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "linux",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

## Create a web app

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "clientCertExclusionPaths": null,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

`https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `DOTNET|5.0`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
'DOTNET|5.0' --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

You've created an empty web app in a Linux container, with git deployment enabled.

#### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

`https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

## Configure connection string

To set connection strings for your Azure app, use the `az webapp config appsettings set` command in the Cloud Shell. In the following command, replace `<app-name>`, as well as the `<connection-string>` parameter with the connection string you created earlier.

```
az webapp config connection-string set --resource-group myResourceGroup --name <app-name> --settings
MyDbConnection='<connection-string>' --connection-string-type SQLAzure
```

In ASP.NET Core, you can use this named connection string (`MyDbConnection`) using the standard pattern, like any connection string specified in `appsettings.json`. In this case, `MyDbConnection` is also defined in your `appsettings.json`. When running in App Service, the connection string defined in App Service takes precedence over the connection string defined in your `appsettings.json`. The code uses the `appsettings.json` value during local development, and the same code uses the App Service value when deployed.

To see how the connection string is referenced in your code, see [Configure app to connect to production database](#).

## Push to Azure from Git

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace

<deploymentLocalGitUrl-from-create-step> with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in **Configure a deployment user**, not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

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

```
Enumerating objects: 268, done.
Counting objects: 100% (268/268), done.
Compressing objects: 100% (171/171), done.
Writing objects: 100% (268/268), 1.18 MiB | 1.55 MiB/s, done.
Total 268 (delta 95), reused 251 (delta 87), pack-reused 0
remote: Resolving deltas: 100% (95/95), done.
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id '64821c3558'.
remote: Generating deployment script.
remote: Project file path: .\DotNetCoreSqlDb.csproj
remote: Generating deployment script for ASP.NET MSBuild16 App
remote: Generated deployment script files
remote: Running deployment command...
remote: Handling ASP.NET Core Web Application deployment with MSBuild16.
remote: .
remote: .
remote: .
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Triggering recycle (preview mode disabled).
remote: App container will begin restart within 10 seconds.
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
 * [new branch] main -> main
```

```
Enumerating objects: 273, done.
Counting objects: 100% (273/273), done.
Delta compression using up to 4 threads
Compressing objects: 100% (175/175), done.
Writing objects: 100% (273/273), 1.19 MiB | 1.85 MiB/s, done.
Total 273 (delta 96), reused 259 (delta 88)
remote: Resolving deltas: 100% (96/96), done.
remote: Deploy Async
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id 'cccecf86c5'.
remote: Repository path is /home/site/repository
remote: Running oryx build...
remote: Build orchestrated by Microsoft Oryx, https://github.com/Microsoft/Oryx
remote: You can report issues at https://github.com/Microsoft/Oryx/issues
remote: .
remote: .
remote: .
remote: Done.
remote: Running post deployment command(s)...
remote: Triggering recycle (preview mode disabled).
remote: Deployment successful.
remote: Deployment Logs : 'https://<app-name>.scm.azurewebsites.net/newui/jsonviewer?view_url=/api/deployments/cccecf86c56493ffa594e76ea1deb3abb3702d89/log'
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
 * [new branch] main -> main
```

## Browse to the Azure app

1. Browse to the deployed app using your web browser.

```
http://<app-name>.azurewebsites.net
```

2. Add a few to-do items.

Description	Created Date	
Deploy app to Azure	2019-08-01	Edit   Details   Delete
Walk dog	2019-08-03	Edit   Details   Delete
Feed cat	2019-08-04	Edit   Details   Delete

Congratulations! You're running a data-driven ASP.NET Core app in App Service.

# Update locally and redeploy

In this step, you make a change to your database schema and publish it to Azure.

## Update your data model

Open *Models/Todo.cs* in the code editor. Add the following property to the `ToDo` class:

```
public bool Done { get; set; }
```

## Rerun database migrations

Run a few commands to make updates to the production database.

```
dotnet ef migrations add AddProperty
dotnet ef database update
```

### NOTE

If you open a new terminal window, you need to set the connection string to the production database in the terminal, like you did in [Run database migrations to the production database](#).

## Use the new property

Make some changes in your code to use the `Done` property. For simplicity in this tutorial, you're only going to change the `Index` and `Create` views to see the property in action.

1. Open *Controllers/TodosController.cs*.
2. Find the `Create([Bind("ID,Description,CreatedDate")] Todo todo)` method and add `Done` to the list of properties in the `Bind` attribute. When you're done, your `Create()` method signature looks like the following code:

```
public async Task<IActionResult> Create([Bind("ID,Description,CreatedDate,Done")] Todo todo)
```

3. Open *Views/Todos/Create.cshtml*.
4. In the Razor code, you should see a `<div class="form-group">` element for `Description`, and then another `<div class="form-group">` element for `CreatedDate`. Immediately following these two elements, add another `<div class="form-group">` element for `Done`:

```
<div class="form-group">
 <label asp-for="Done" class="col-md-2 control-label"></label>
 <div class="col-md-10">
 <input asp-for="Done" class="form-control" />

 </div>
</div>
```

5. Open *Views/Todos/Index.cshtml*.
6. Search for the empty `<th></th>` element. Just above this element, add the following Razor code:

```
<th>
 @Html.DisplayNameFor(model => model.Done)
</th>
```

7. Find the `<td>` element that contains the `asp-action` tag helpers. Just above this element, add the following Razor code:

```
<td>
 @Html.DisplayFor(modelItem => item.Done)
</td>
```

That's all you need to see the changes in the `Index` and `Create` views.

## Test your changes locally

1. Run the app locally.

```
dotnet run
```

### NOTE

If you open a new terminal window, you need to set the connection string to the production database in the terminal, like you did in [Run database migrations to the production database](#).

2. In your browser, navigate to `http://localhost:5000/`. You can now add a to-do item and check **Done**. Then it should show up in your homepage as a completed item. Remember that the `Edit` view doesn't show the `Done` field, because you didn't change the `Edit` view.

## Publish changes to Azure

1. Commit your changes to Git and push it to your App Service app.

```
git add .
git commit -m "added done field"
git push azure main
```

2. Once the `git push` is complete, navigate to your App Service app and try adding a to-do item and check **Done**.

Description	Created Date	Done
Deploy app to Azure	2019-08-01	<input type="checkbox"/>
Walk dog	2019-08-03	<input type="checkbox"/>
Feed cat	2019-08-04	<input type="checkbox"/>
Check email	2019-08-05	<input checked="" type="checkbox"/>

All your existing to-do items are still displayed. When you republish your ASP.NET Core app, existing data in your SQL Database isn't lost. Also, Entity Framework Core Migrations only changes the data schema and leaves your existing data intact.

## Stream diagnostic logs

While the ASP.NET Core app runs in Azure App Service, you can get the console logs piped to the Cloud Shell. That way, you can get the same diagnostic messages to help you debug application errors.

The sample project already follows the guidance for the [Azure App Service logging provider](#) with two configuration changes:

- Includes a reference to `Microsoft.Extensions.Logging.AzureAppServices` in `DotNetCoreSqlDb.csproj`.
- Calls `loggerFactory.AddAzureWebAppDiagnostics()` in `Program.cs`.

- To set the ASP.NET Core [log level](#) in App Service to `Information` from the default level `Error`, use the `az webapp log config` command in the Cloud Shell.

```
az webapp log config --name <app-name> --resource-group myResourceGroup --application-logging filesystem --level information
```

### NOTE

The project's log level is already set to `Information` in `appsettings.json`.

- To start log streaming, use the `az webapp log tail` command in the Cloud Shell.

```
az webapp log tail --name <app-name> --resource-group myResourceGroup
```

- Once log streaming has started, refresh the Azure app in the browser to get some web traffic. You can now see console logs piped to the terminal. If you don't see console logs immediately, check again in 30 seconds.

4. To stop log streaming at any time, type `Ctrl + C`.

For more information on customizing the ASP.NET Core logs, see [Logging in .NET](#).

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

What you learned:

- Create a SQL Database in Azure
- Connect a ASP.NET Core app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to map a custom DNS name to your app.

[Tutorial: Map custom DNS name to your app](#)

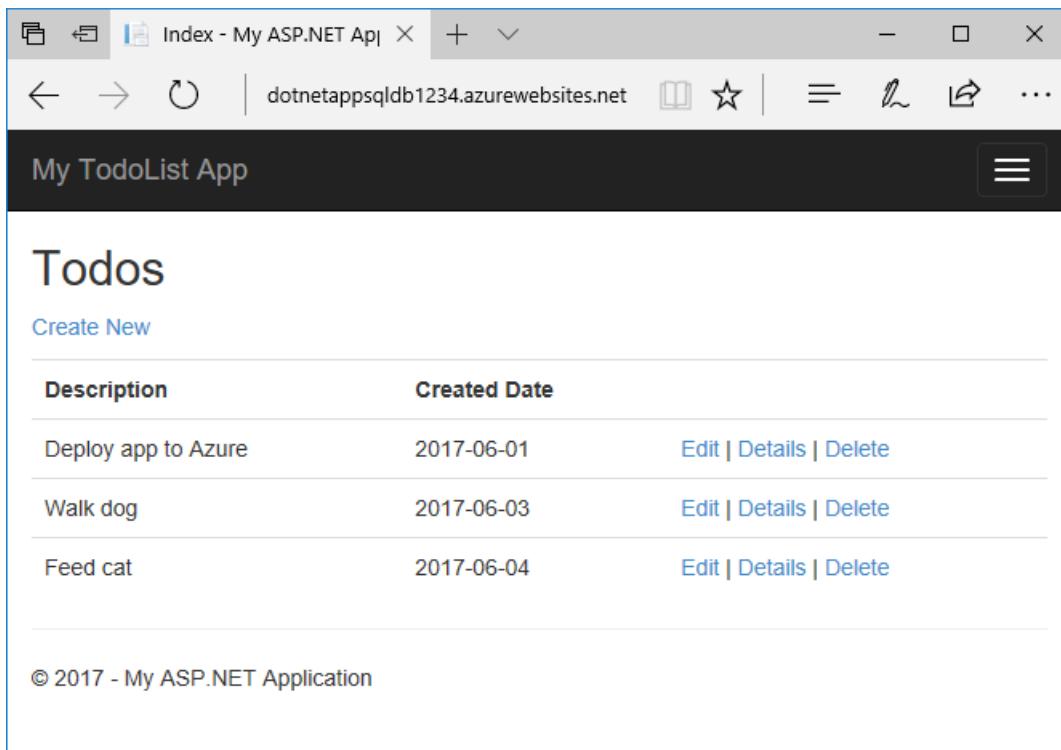
Or, check out other resources:

[Configure ASP.NET Core app](#)

# Tutorial: Deploy an ASP.NET app to Azure with Azure SQL Database

11/2/2021 • 12 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows you how to deploy a data-driven ASP.NET app in App Service and connect it to [Azure SQL Database](#). When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database.



In this tutorial, you learn how to:

- Create a database in Azure SQL Database
- Connect an ASP.NET app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial:

Install [Visual Studio 2019](#) with the [ASP.NET and web development workload](#).

If you've installed Visual Studio already, add the workloads in Visual Studio by clicking **Tools > Get Tools and Features**.

[Download the sample](#)

1. Download the sample project.
2. Extract (unzip) the `dotnet-sqldb-tutorial-master.zip` file.

The sample project contains a basic [ASP.NET MVC](#) create-read-update-delete (CRUD) app using [Entity Framework Code First](#).

### Run the app

1. Open the `dotnet-sqldb-tutorial-master/DotNetAppSqlDb.sln` file in Visual Studio.
2. Type `Ctrl+F5` to run the app without debugging. The app is displayed in your default browser.
3. Select the **Create New** link and create a couple *to-do* items.

The screenshot shows a web browser window with the title "Index - My ASP.NET App". The address bar shows "localhost:1234". The main content area is titled "My TodoList App" and contains a section titled "Todos". Below this is a table with two rows:

Description	Created Date	
Walk dog	2017-06-01	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Build Azure ASP.NET app	2017-06-04	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, there is a copyright notice: "© 2017 - My ASP.NET Application".

4. Test the **Edit**, **Details**, and **Delete** links.

The app uses a database context to connect with the database. In this sample, the database context uses a connection string named `MyDbContext`. The connection string is set in the `Web.config` file and referenced in the `Models/MyDatabaseContext.cs` file. The connection string name is used later in the tutorial to connect the Azure app to an Azure SQL Database.

## Publish ASP.NET application to Azure

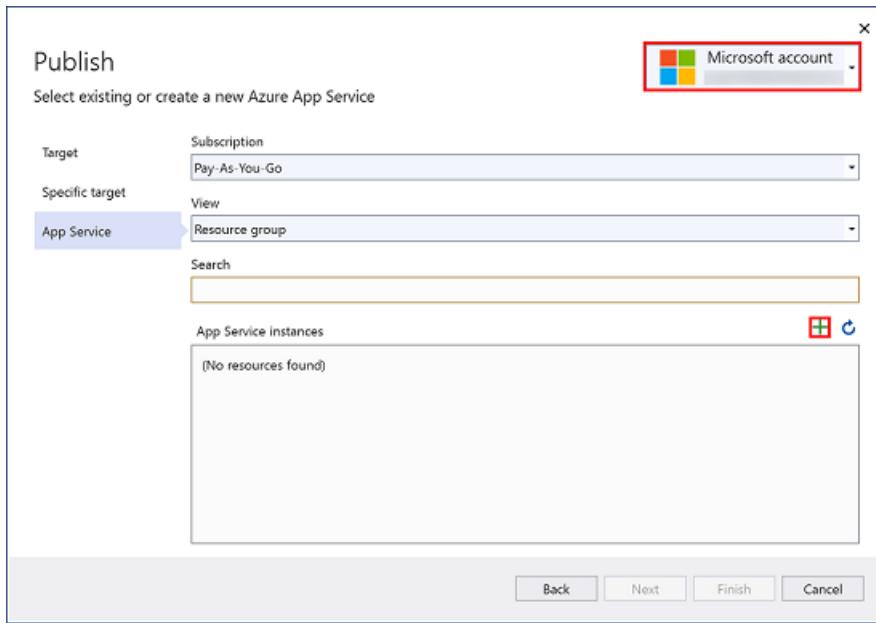
1. In the **Solution Explorer**, right-click your `DotNetAppSqlDb` project and select **Publish**.

The screenshot shows the Visual Studio **Solution Explorer** window. A project named "DotNetAppSqlDb" is selected. In the context menu (which is open), the "Publish..." option is highlighted with a red box. Other options like "Build", "Rebuild", "Clean", and "Analyze" are also visible in the menu.

2. Select Azure as your target and click **Next**.
3. Make sure that **Azure App Service (Windows)** is selected and click **Next**.

#### Sign in and add an app

1. In the **Publish** dialog, click **Sign In**.
2. Sign in to your Azure subscription. If you're already signed into a Microsoft account, make sure that account holds your Azure subscription. If the signed-in Microsoft account doesn't have your Azure subscription, click it to add the correct account.
3. In the **App Service instances** pane, click **+**.

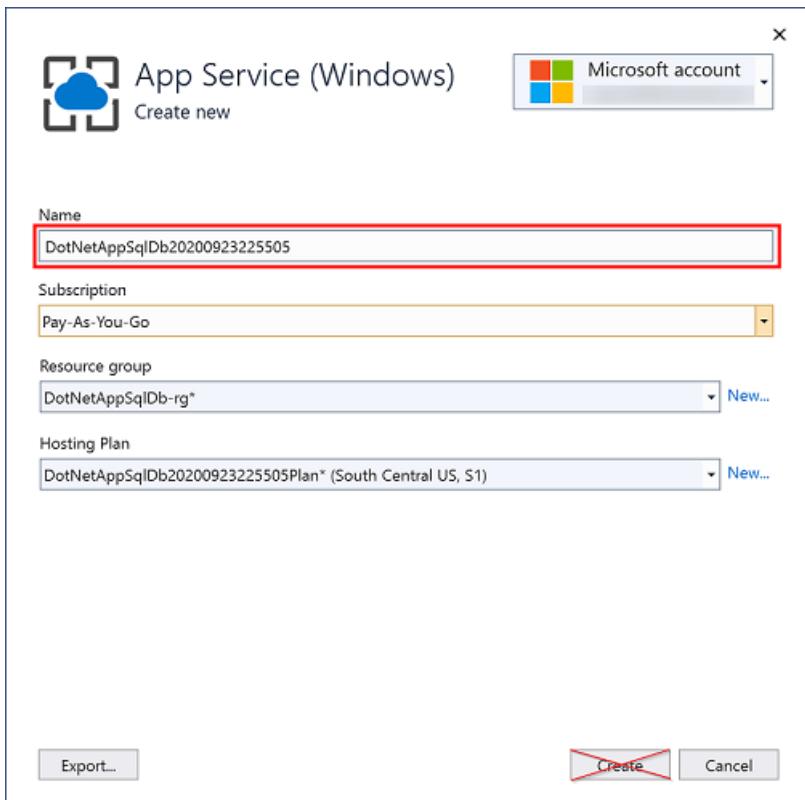


#### Configure the web app name

You can keep the generated web app name, or change it to another unique name (valid characters are `a-z`, `0-9`, and `-`). The web app name is used as part of the default URL for your app (`<app_name>.azurewebsites.net`, where `<app_name>` is your web app name). The web app name needs to be unique across all apps in Azure.

#### NOTE

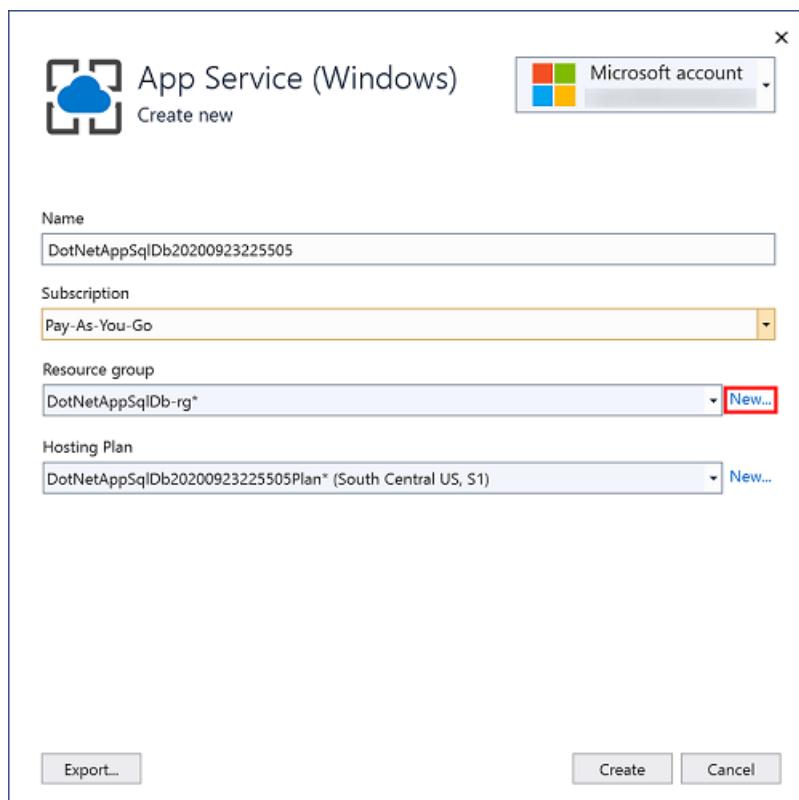
Don't select **Create** yet.



#### Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

1. Next to Resource Group, click New.



2. Name the resource group **myResourceGroup**.

#### Create an App Service plan

An [App Service plan](#) specifies the location, size, and features of the web server farm that hosts your app. You can save money when you host multiple apps by configuring the web apps to share a single App Service plan.

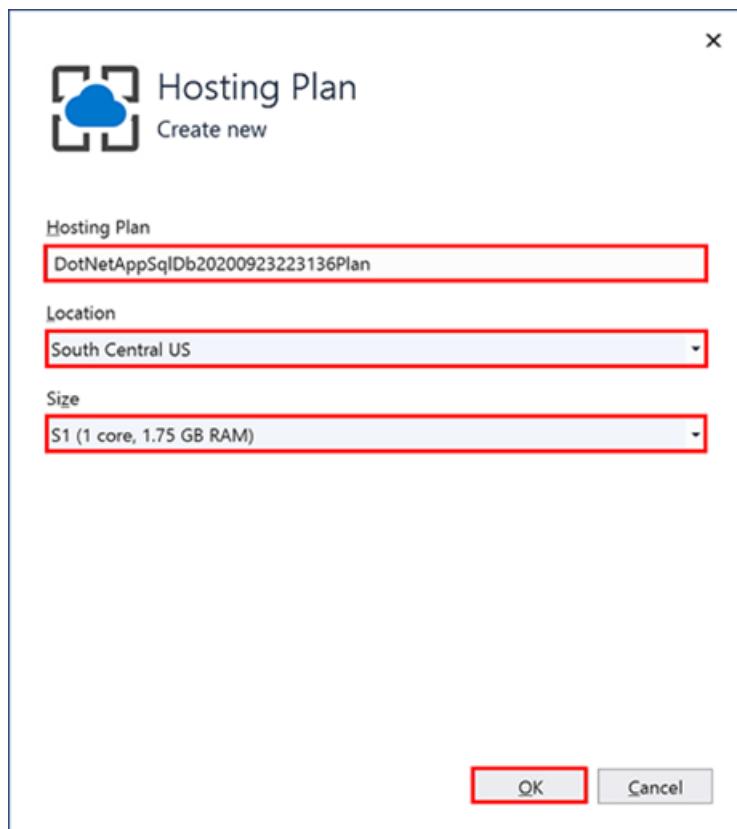
App Service plans define:

- Region (for example: North Europe, East US, or Southeast Asia)
- Instance size (small, medium, or large)
- Scale count (1 to 20 instances)
- SKU (Free, Shared, Basic, Standard, or Premium)

1. Next to **Hosting Plan**, click **New**.

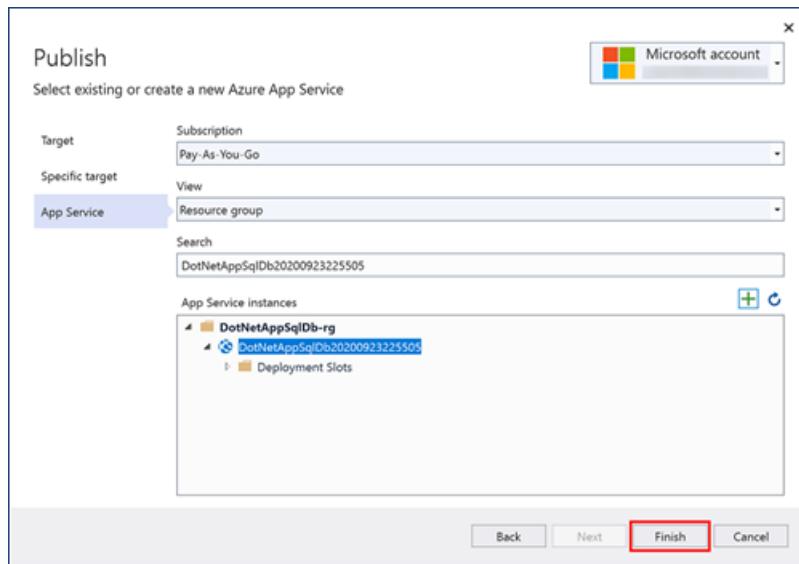
2. In the **Configure App Service Plan** dialog, configure the new App Service plan with the following settings and click **OK**:

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
App Service Plan	myAppServicePlan	<a href="#">App Service plans</a>
Location	West Europe	<a href="#">Azure regions</a>
Size	Free	<a href="#">Pricing tiers</a>



3. Click **Create** and wait for the Azure resources to be created.

4. The **Publish** dialog shows the resources you've configured. Click **Finish**.



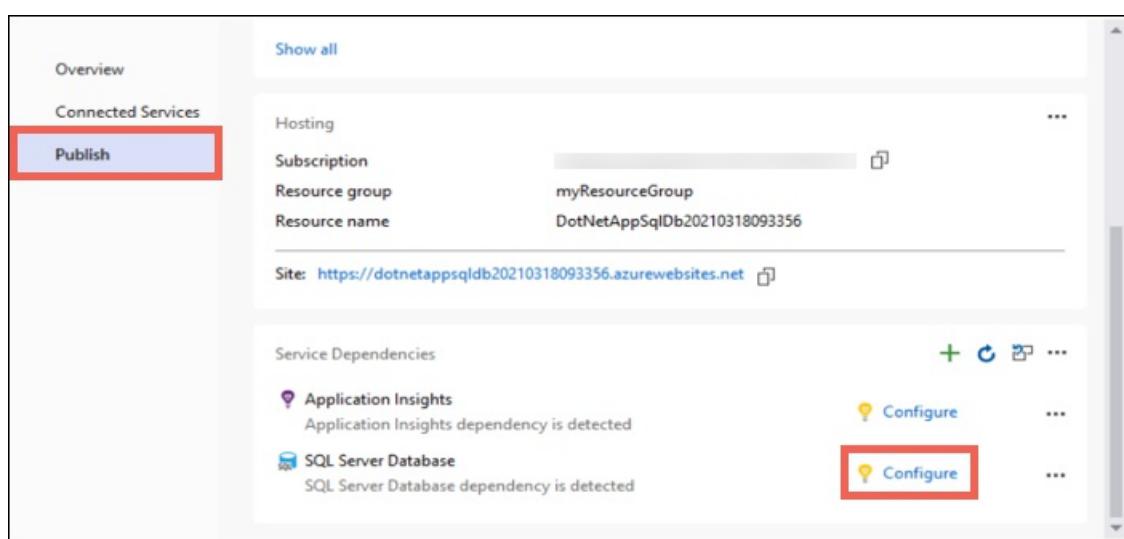
#### Create a server and database

Before creating a database, you need a [logical SQL server](#). A logical SQL server is a logical construct that contains a group of databases managed as a group.

1. In the **Publish** dialog, scroll down to the **Service Dependencies** section. Next to **SQL Server Database**, click **Configure**.

#### NOTE

Be sure to configure the SQL Database from the Publish page instead of the Connected Services page.



2. Select **Azure SQL Database** and click **Next**.
3. In the **Configure Azure SQL Database** dialog, click **+**.
4. Next to **Database server**, click **New**.

The server name is used as part of the default URL for your server, `<server_name>.database.windows.net`. It must be unique across all servers in Azure SQL. Change the server name to a value you want.

5. Add an administrator username and password. For password complexity requirements, see [Password Policy](#).

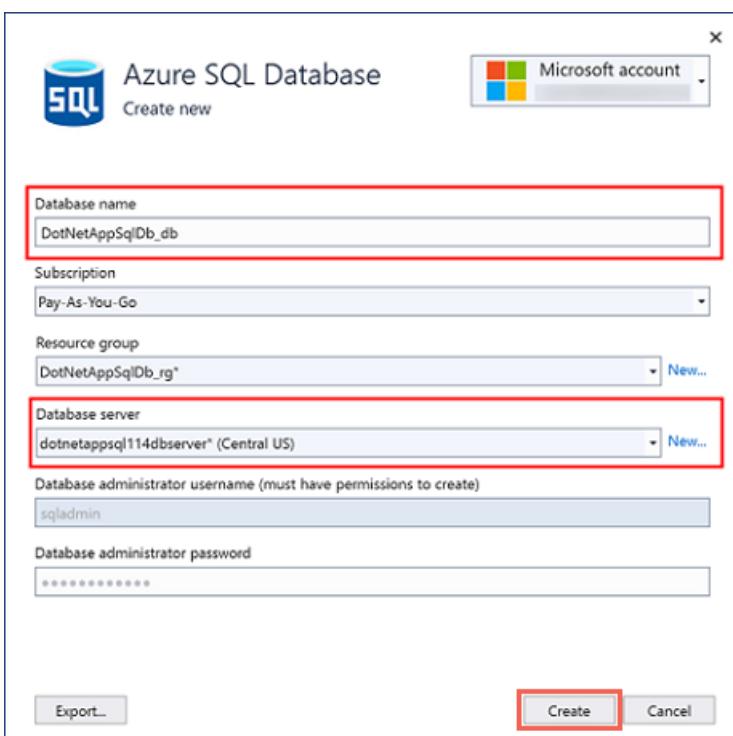
Remember this username and password. You need them to manage the server later.



#### IMPORTANT

Even though your password in the connection strings is masked (in Visual Studio and also in App Service), the fact that it's maintained somewhere adds to the attack surface of your app. App Service can use [managed service identities](#) to eliminate this risk by removing the need to maintain secrets in your code or app configuration at all. For more information, see [Next steps](#).

6. Click OK.
7. In the Azure SQL Database dialog, keep the default generated Database Name. Select Create and wait for the database resources to be created.

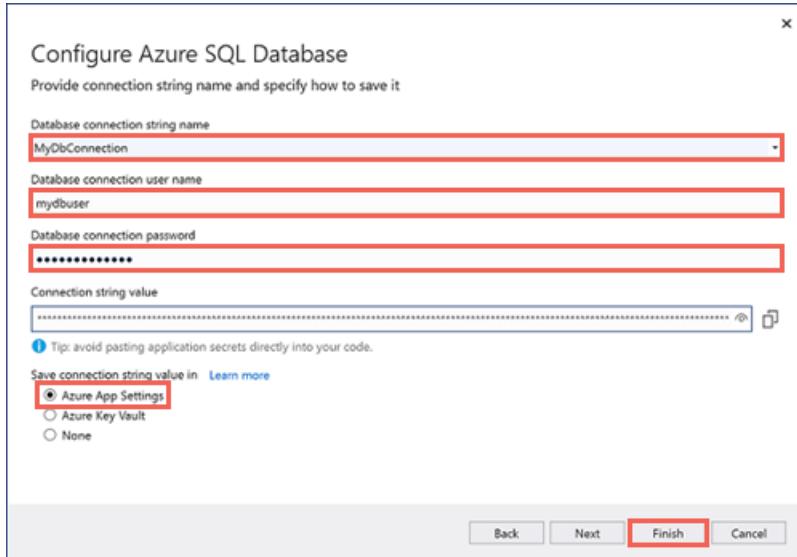


#### Configure database connection

1. When the wizard finishes creating the database resources, click **Next**.
2. In the **Database connection string Name**, type *MyDbConnection*. This name must match the connection string that is referenced in *Models/MyDbContext.cs*.
3. In **Database connection user name** and **Database connection password**, type the administrator username and password you used in [Create a server](#).
4. Make sure **Azure App Settings** is selected and click **Finish**.

#### NOTE

If you see **Local user secrets files** instead, you must have configured SQL Database from the **Connected Services** page instead of the **Publish** page.



5. Wait for configuration wizard to finish and click **Close**.

#### Deploy your ASP.NET app

1. In the **Publish** tab scroll back up to the top and click **Publish**. Once your ASP.NET app is deployed to Azure. Your default browser is launched with the URL to the deployed app.
2. Add a few to-do items.

My TodoList App

## Todos

Create New

Description	Created Date	
Deploy app to Azure	2017-06-01	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Walk dog	2017-06-03	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Feed cat	2017-06-04	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2017 - My ASP.NET Application

Congratulations! Your data-driven ASP.NET application is running live in Azure App Service.

## Access the database locally

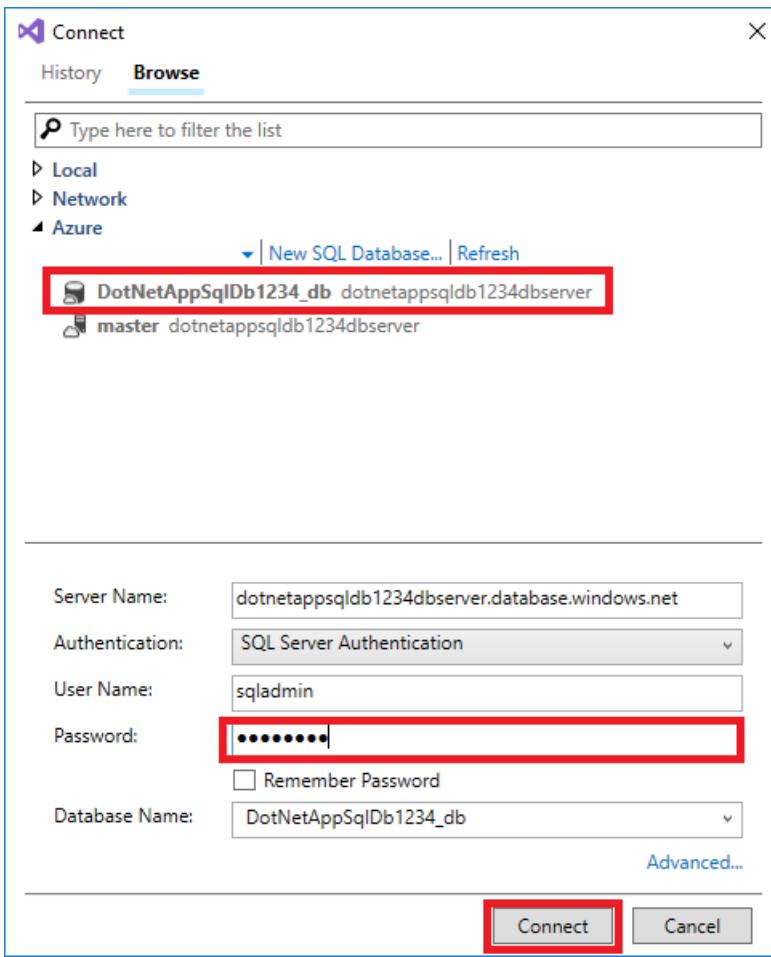
Visual Studio lets you explore and manage your new database in Azure easily in the **SQL Server Object Explorer**. The new database already opened its firewall to the App Service app that you created, but to access it from your local computer (such as from Visual Studio), you must open a firewall for your local machine's public IP address. If your internet service provider changes your public IP address, you need to reconfigure the firewall to access the Azure database again.

### Create a database connection

1. From the View menu, select **SQL Server Object Explorer**.
2. At the top of **SQL Server Object Explorer**, click the **Add SQL Server** button.

### Configure the database connection

1. In the **Connect** dialog, expand the **Azure** node. All your SQL Database instances in Azure are listed here.
2. Select the database that you created earlier. The connection you created earlier is automatically filled at the bottom.
3. Type the database administrator password you created earlier and click **Connect**.

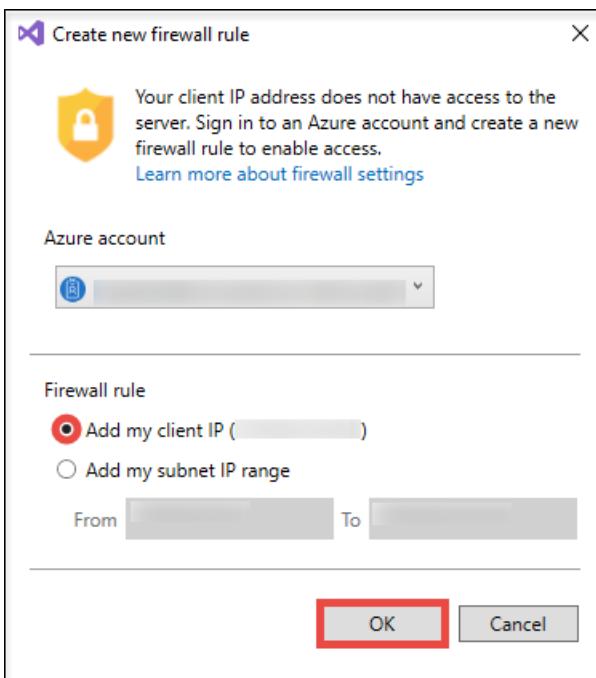


#### Allow client connection from your computer

The **Create a new firewall rule** dialog is opened. By default, a server only allows connections to its databases from Azure services, such as your Azure app. To connect to your database from outside of Azure, create a firewall rule at the server level. The firewall rule allows the public IP address of your local computer.

The dialog is already filled with your computer's public IP address.

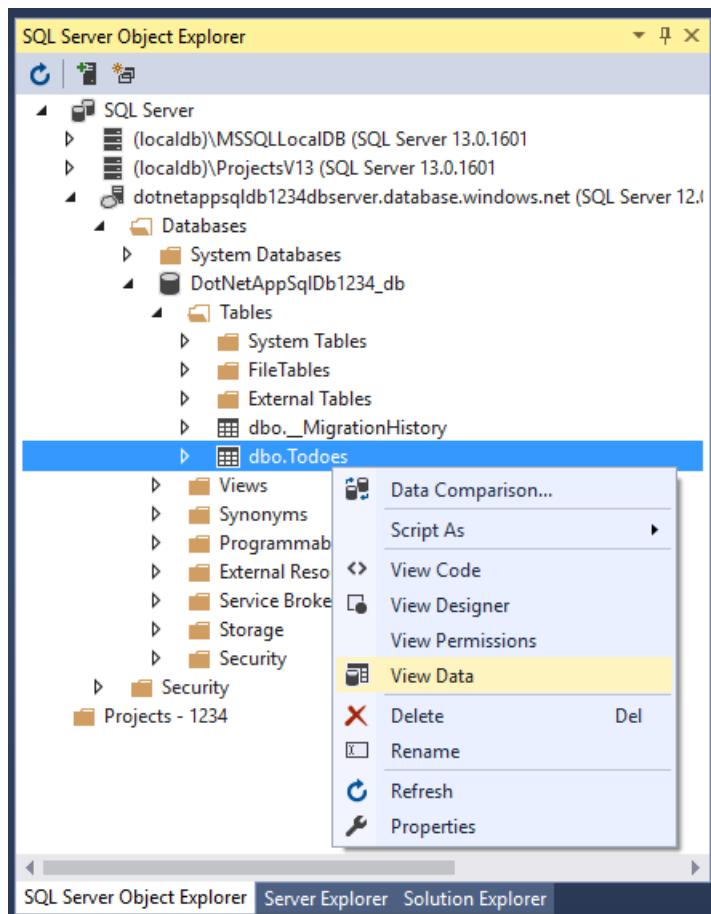
1. Make sure that **Add my client IP** is selected and click **OK**.



Once Visual Studio finishes creating the firewall setting for your SQL Database instance, your connection shows up in **SQL Server Object Explorer**.

Here, you can perform the most common database operations, such as run queries, create views and stored procedures, and more.

2. Expand your connection > **Databases** > <your database> > **Tables**. Right-click on the `Todos` table and select **View Data**.



## Update app with Code First Migrations

You can use the familiar tools in Visual Studio to update your database and app in Azure. In this step, you use Code First Migrations in Entity Framework to make a change to your database schema and publish it to Azure.

For more information about using Entity Framework Code First Migrations, see [Getting Started with Entity Framework 6 Code First using MVC 5](#).

### Update your data model

Open `Models\Todo.cs` in the code editor. Add the following property to the `Todo` class:

```
public bool Done { get; set; }
```

### Run Code First Migrations locally

Run a few commands to make updates to your local database.

1. From the Tools menu, click **NuGet Package Manager** > **Package Manager Console**.
2. In the Package Manager Console window, enable Code First Migrations:

```
Enable-Migrations
```

3. Add a migration:

```
Add-Migration AddProperty
```

4. Update the local database:

```
Update-Database
```

5. Type `Ctrl+F5` to run the app. Test the edit, details, and create links.

If the application loads without errors, then Code First Migrations has succeeded. However, your page still looks the same because your application logic is not using this new property yet.

#### Use the new property

Make some changes in your code to use the `Done` property. For simplicity in this tutorial, you're only going to change the `Index` and `Create` views to see the property in action.

1. Open `Controllers\TodosController.cs`.

2. Find the `Create()` method on line 52 and add `Done` to the list of properties in the `Bind` attribute. When you're done, your `Create()` method signature looks like the following code:

```
public ActionResult Create([Bind(Include = "Description,CreatedDate,Done")] Todo todo)
```

3. Open `Views\Todos\Create.cshtml`.

4. In the Razor code, you should see a `<div class="form-group">` element that uses `model.Description`, and then another `<div class="form-group">` element that uses `model.CreatedDate`. Immediately following these two elements, add another `<div class="form-group">` element that uses `model.Done`:

```
<div class="form-group">
 @Html.LabelFor(model => model.Done, htmlAttributes: new { @class = "control-label col-md-2" })
 <div class="col-md-10">
 <div class="checkbox">
 @Html.EditorFor(model => model.Done)
 @Html.ValidationMessageFor(model => model.Done, "", new { @class = "text-danger" })
 </div>
 </div>
</div>
```

5. Open `Views\Todos\Index.cshtml`.

6. Search for the empty `<th></th>` element. Just above this element, add the following Razor code:

```
<th>
 @Html.DisplayNameFor(model => model.Done)
</th>
```

7. Find the `<td>` element that contains the `Html.ActionLink()` helper methods. *Above* this `<td>`, add another `<td>` element with the following Razor code:

```
<td>
 @Html.DisplayFor(modelItem => item.Done)
</td>
```

That's all you need to see the changes in the `Index` and `Create` views.

8. Type `Ctrl+F5` to run the app.

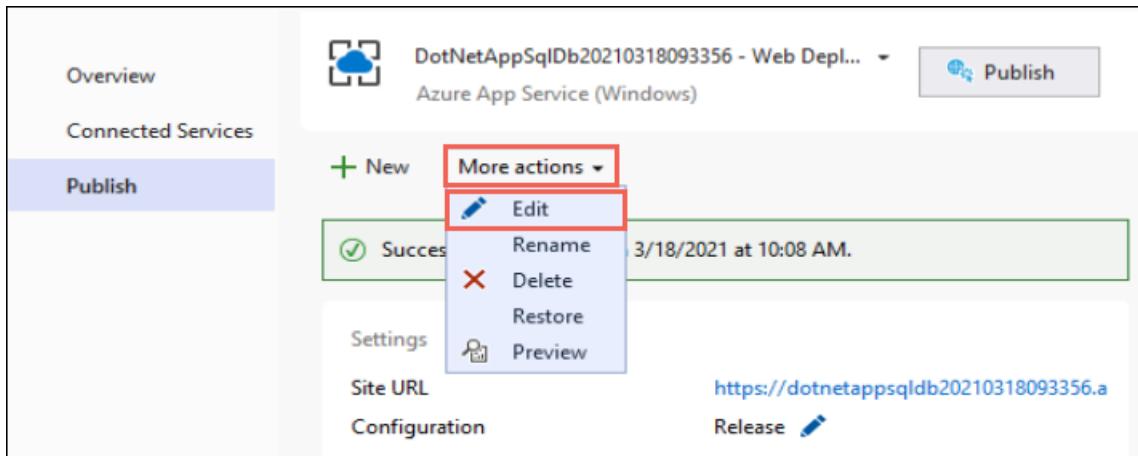
You can now add a to-do item and check **Done**. Then it should show up in your homepage as a completed item. Remember that the `Edit` view doesn't show the `Done` field, because you didn't change the `Edit` view.

#### Enable Code First Migrations in Azure

Now that your code change works, including database migration, you publish it to your Azure app and update your SQL Database with Code First Migrations too.

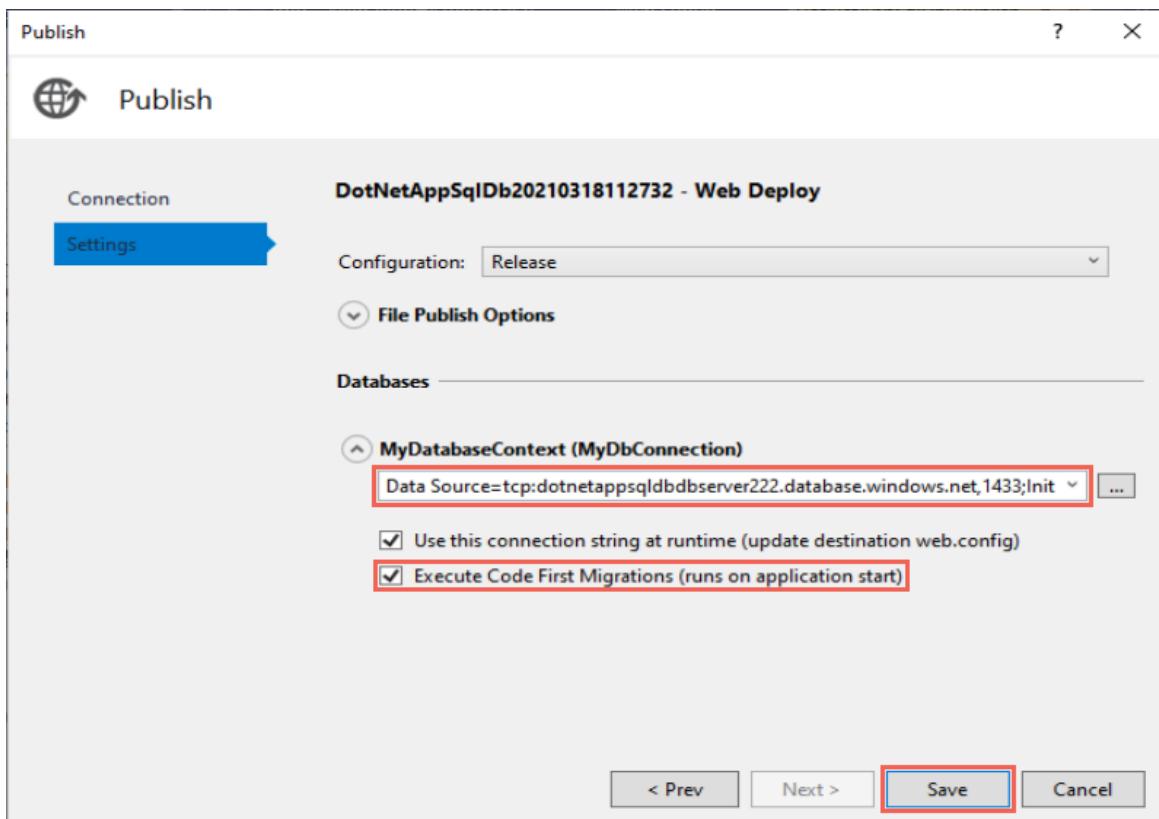
1. Just like before, right-click your project and select **Publish**.

2. Click **More actions > Edit** to open the publish settings.



3. In the **MyDbContext** dropdown, select the database connection for your Azure SQL Database.

4. Select **Execute Code First Migrations (runs on application start)**, then click **Save**.



#### Publish your changes

Now that you enabled Code First Migrations in your Azure app, publish your code changes.

1. In the publish page, click **Publish**.

2. Try adding to-do items again and select **Done**, and they should show up in your homepage as a completed item.

The screenshot shows a web browser window with the title "Index - My ASP.NET App". The URL in the address bar is "dotnetappsqldb1234.azurewebsites.net". The page content is titled "My TodoList App" and "Todos". It displays a table with three rows:

Description	Created Date	Done	Action
Deploy app to Azure	2017-06-01	<input checked="" type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Walk dog	2017-06-03	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Feed cat	2017-06-04	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

At the bottom of the page, there is a copyright notice: "© 2017 - My ASP.NET Application".

All your existing to-do items are still displayed. When you republish your ASP.NET application, existing data in your SQL Database is not lost. Also, Code First Migrations only changes the data schema and leaves your existing data intact.

## Stream application logs

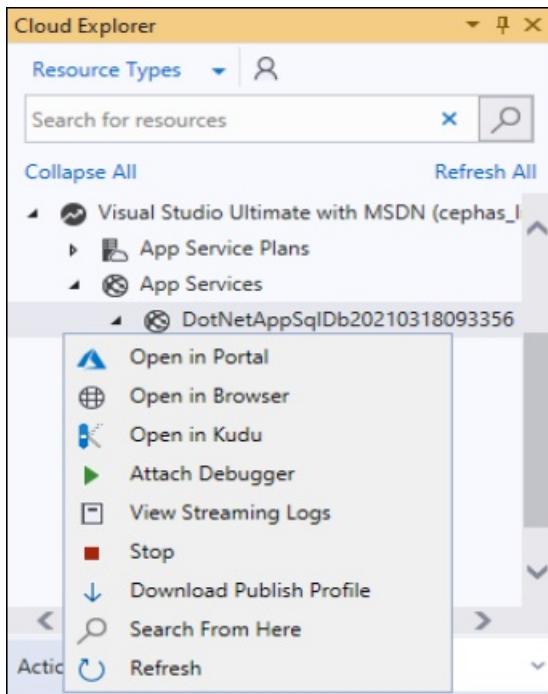
You can stream tracing messages directly from your Azure app to Visual Studio.

Open *Controllers\TodosController.cs*.

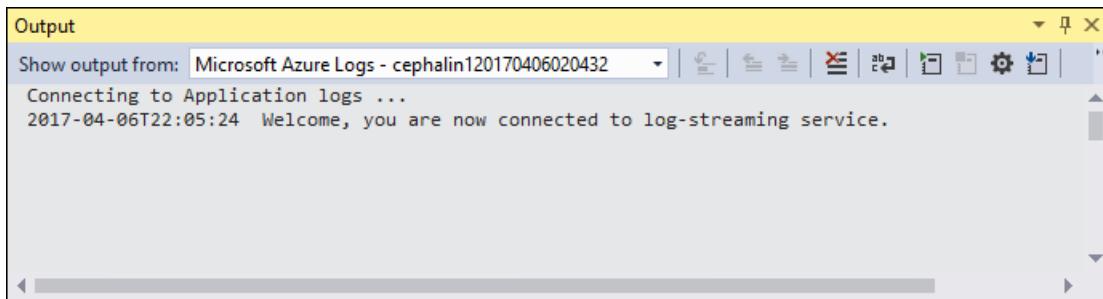
Each action starts with a `Trace.WriteLine()` method. This code is added to show you how to add trace messages to your Azure app.

### Enable log streaming

1. From the **View** menu, select **Cloud Explorer**.
2. In **Cloud Explorer**, expand the Azure subscription that has your app and expand **App Service**.
3. Right-click your Azure app and select **View Streaming Logs**.



The logs are now streamed into the **Output** window.



However, you don't see any of the trace messages yet. That's because when you first select **View Streaming Logs**, your Azure app sets the trace level to `Error`, which only logs error events (with the `Trace.TraceError()` method).

#### Change trace levels

1. To change the trace levels to output other trace messages, go back to **Cloud Explorer**.
2. Right-click your app again and select **Open in Portal**.
3. In the portal management page for your app, from the left menu, select **App Service logs**.
4. Under **Application Logging (File System)**, select **Verbose** in **Level**. Click **Save**.

**Actions**

- Open in Management Portal
- Stop Web App
- Restart Web App

**Web App Settings** [Learn more](#)

.NET Framework Version	v4.5
Web Server Logging	Off
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Verbose
Remote Debugging	On

#### TIP

You can experiment with different trace levels to see what types of messages are displayed for each level. For example, the **Information** level includes all logs created by `Trace.TraceInformation()`, `Trace.TraceWarning()`, and `Trace.TraceError()`, but not logs created by `Trace.WriteLine()`.

5. In your browser navigate to your app again at `http://<your app name>.azurewebsites.net`, then try clicking around the to-do list application in Azure. The trace messages are now streamed to the **Output** window in Visual Studio.

```
Application: 2017-04-06T23:30:41 PID[8132] Verbose GET /Todos/Index
Application: 2017-04-06T23:30:43 PID[8132] Verbose GET /Todos/Create
Application: 2017-04-06T23:30:53 PID[8132] Verbose POST /Todos/Create
Application: 2017-04-06T23:30:54 PID[8132] Verbose GET /Todos/Index
```

#### Stop log streaming

To stop the log-streaming service, click the **Stop monitoring** button in the **Output** window.

**Output**

Show output from: Microsoft Azure Logs - DotNetAppSqlDb1234

Connecting to Application logs ...

2017-06-02T02:36:59 Welcome, you are now connected

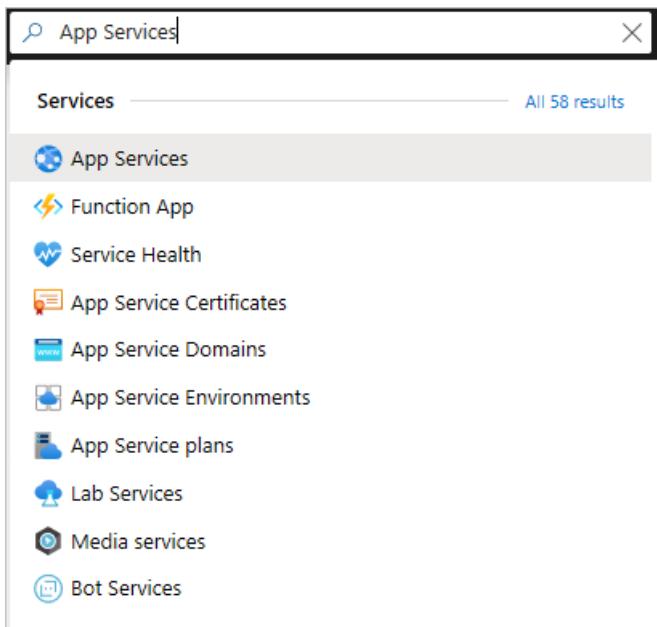
Stop monitoring real time Microsoft Azure logs

```
Application: 2017-06-02T02:37:20 PID[9092] Verbose GET /Todos/Delete/1
Application: 2017-06-02T02:37:29 PID[9092] Verbose GET /Todos/Edit/1
Application: 2017-06-02T02:37:41 PID[9092] Verbose POST /Todos/Edit/1
Application: 2017-06-02T02:37:42 PID[9092] Verbose GET /Todos/Index
Application: 2017-06-02T02:37:48 PID[9092] Verbose GET /Todos/Delete/3
Application: 2017-06-02T02:37:51 PID[9092] Verbose POST /Todos/Delete/3
Application: 2017-06-02T02:37:52 PID[9092] Verbose GET /Todos/Index
Application: 2017-06-02T02:38:59 No new trace in the past 1 min(s).
Application: 2017-06-02T02:39:59 No new trace in the past 2 min(s).
Application: 2017-06-02T02:40:59 No new trace in the past 3 min(s).
```

Package Manager Console Error List Data Tools Operations Web Publish Activity **Output**

## Manage your Azure app

Go to the [Azure portal](#) to manage the web app. Search for and select **App Services**.



Select the name of your Azure app.

A screenshot of the Azure App Services list view. At the top, there are buttons for "Add", "Columns", and "Refresh". Below that, it says "Subscriptions: All 2 selected". There are four filter dropdowns: "Filter by name...", "All subscriptions", "All locations", and "No grouping". A message indicates "1 items". The table has columns: NAME, RESOURCE GR..., STATUS, APP TYPE, and APP SERVICE PLAN. A single row is shown, representing the app "DotNetAppSqlDb1234", which is highlighted with a red border. The details for this app are: myResourceGroup, Running, Web app, myAppServicePlan, and three vertical ellipsis (...).

You have landed in your app's page.

By default, the portal shows the **Overview** page. This page gives you a view of how your app is doing. Here, you can also perform basic management tasks like browse, stop, start, restart, and delete. The tabs on the left side of the page show the different configuration pages you can open.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, you can delete them by deleting the resource group.

1. From your web app's **Overview** page in the Azure portal, select the **myResourceGroup** link under **Resource group**.
2. On the resource group page, make sure that the listed resources are the ones you want to delete.
3. Select **Delete**, type **myResourceGroup** in the text box, and then select **Delete**.

## Next steps

In this tutorial, you learned how to:

- Create a database in Azure SQL Database
- Connect an ASP.NET app to SQL Database
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to easily improve the security of your connection Azure SQL Database.

[Access SQL Database securely using managed identities for Azure resources](#)

More resources:

[Configure ASP.NET app](#)

Want to optimize and save on your cloud spending?

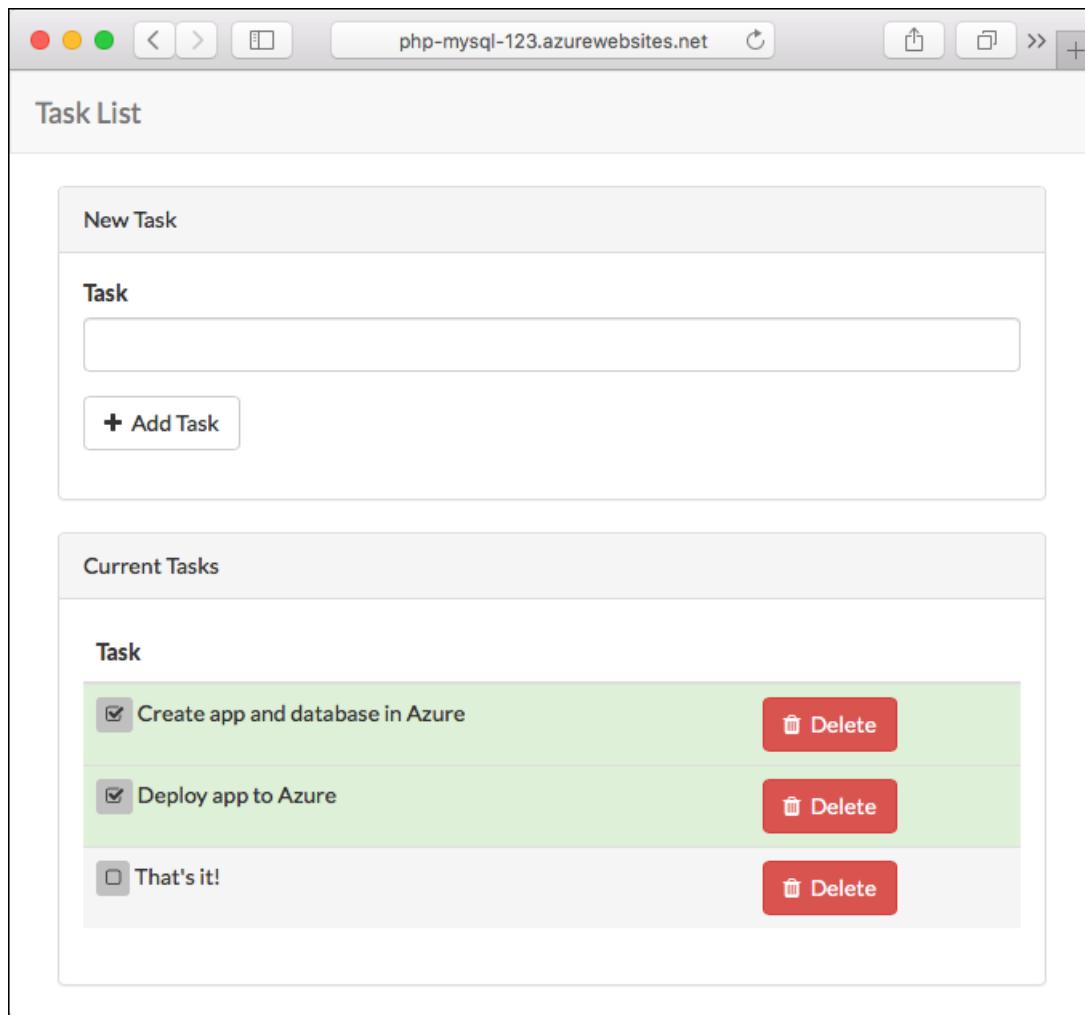
[Start analyzing costs with Cost Management](#)

# Tutorial: Build a PHP and MySQL app in Azure App Service

11/2/2021 • 21 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service using the Windows operating system. This tutorial shows how to create a PHP app in Azure and connect it to a MySQL database. When you're finished, you'll have a [Laravel](#) app running on Azure App Service on Windows.

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create a PHP app in Azure and connect it to a MySQL database. When you're finished, you'll have a [Laravel](#) app running on Azure App Service on Linux.



In this tutorial, you learn how to:

- Create a MySQL database in Azure
- Connect a PHP app to MySQL
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

# Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install PHP 5.6.4 or above](#)
- [Install Composer](#)
- Enable the following PHP extensions Laravel needs: OpenSSL, PDO-MySQL, Mbstring, Tokenizer, XML
- [Install and start MySQL](#)
- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

## Prepare local MySQL

In this step, you create a database in your local MySQL server for your use in this tutorial.

### Connect to local MySQL server

In a terminal window, connect to your local MySQL server. You can use this terminal window to run all the commands in this tutorial.

```
mysql -u root -p
```

If you're prompted for a password, enter the password for the `root` account. If you don't remember your root account password, see [MySQL: How to Reset the Root Password](#).

If your command runs successfully, then your MySQL server is running. If not, make sure that your local MySQL server is started by following the [MySQL post-installation steps](#).

### Create a database locally

1. At the `mysql` prompt, create a database.

```
CREATE DATABASE sampledb;
```

2. Exit your server connection by typing `quit`.

```
quit
```

## Create a PHP app locally

In this step, you get a Laravel sample application, configure its database connection, and run it locally.

## Clone the sample

In the terminal window, `cd` to a working directory.

1. Clone the sample repository and change to the repository root.

```
git clone https://github.com/Azure-Samples/laravel-tasks
cd laravel-tasks
```

2. Make sure the default branch is `main`.

```
git branch -m main
```

### TIP

The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main`, this tutorial also shows you how to deploy a repository from `main`. For more information, see [Change deployment branch](#).

3. Install the required packages.

```
composer install
```

## Configure MySQL connection

In the repository root, create a file named `.env`. Copy the following variables into the `.env` file. Replace the `<root_password>` placeholder with the MySQL root user's password.

```
APP_ENV=local
APP_DEBUG=true
APP_KEY=

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_DATABASE=sampledbs
DB_USERNAME=root
DB_PASSWORD=<root_password>
```

For information on how Laravel uses the `.env` file, see [Laravel Environment Configuration](#).

## Run the sample locally

1. Run [Laravel database migrations](#) to create the tables the application needs. To see which tables are created in the migrations, look in the `database/migrations` directory in the Git repository.

```
php artisan migrate
```

2. Generate a new Laravel application key.

```
php artisan key:generate
```

3. Run the application.

```
php artisan serve
```

4. Navigate to `http://localhost:8000` in a browser. Add a few tasks in the page.

The screenshot shows a web application titled "Task List" running on "localhost". The interface is divided into two main sections: "New Task" and "Current Tasks".

**New Task:** Contains a text input field and a button labeled "+ Add Task".

**Current Tasks:** Contains three items, each with a "Delete" button:

- Create app and database in Azure
- Deploy data-driven app
- That's it!

5. To stop PHP, type `ctrl + c` in the terminal.

## Create MySQL in Azure

In this step, you create a MySQL database in [Azure Database for MySQL](#). Later, you configure the PHP application to connect to this database.

### Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service in **Free** tier, run the `az appservice list-locations --sku FREE` command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

### Create a MySQL server

In the Cloud Shell, create a server in Azure Database for MySQL with the `az mysql server create` command.

In the following command, substitute a unique server name for the `<mysql-server-name>` placeholder, a user name for the `<admin-user>`, and a password for the `<admin-password>` placeholder. The server name is used as part of your MySQL endpoint (`https://<mysql-server-name>.mysql.database.azure.com`), so the name needs to be unique across all servers in Azure. For details on selecting MySQL DB SKU, see [Create an Azure Database for MySQL server](#).

```
az mysql server create --resource-group myResourceGroup --name <mysql-server-name> --location "West Europe"
--admin-user <admin-user> --admin-password <admin-password> --sku-name B_Gen5_1
```

When the MySQL server is created, the Azure CLI shows information similar to the following example:

```
{
 "administratorLogin": "<admin-user>",
 "administratorLoginPassword": null,
 "fullyQualifiedDomainName": "<mysql-server-name>.mysql.database.azure.com",
 "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/<mysql-
server-name>",
 "location": "westeurope",
 "name": "<mysql-server-name>",
 "resourceGroup": "myResourceGroup",
 ...
- < Output has been truncated for readability >
}
```

## Configure server firewall

1. In the Cloud Shell, create a firewall rule for your MySQL server to allow client connections by using the `az mysql server firewall-rule create` command. When both starting IP and end IP are set to 0.0.0.0, the firewall is only opened for other Azure resources.

```
az mysql server firewall-rule create --name allAzureIPs --server <mysql-server-name> --resource-group
myResourceGroup --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0
```

### TIP

You can be even more restrictive in your firewall rule by [using only the outbound IP addresses your app uses](#).

2. In the Cloud Shell, run the command again to allow access from your local computer by replacing `<your-ip-address>` with [your local IPv4 IP address](#).

```
az mysql server firewall-rule create --name AllowLocalClient --server <mysql-server-name> --resource-
group myResourceGroup --start-ip-address=<your-ip-address> --end-ip-address=<your-ip-address>
```

## Create a production database

1. In the local terminal window, connect to the MySQL server in Azure. Use the value you specified previously for `<admin-user>` and `<mysql-server-name>`. When prompted for a password, use the password you specified when you created the database in Azure.

```
mysql -u <admin-user>@<mysql-server-name> -h <mysql-server-name>.mysql.database.azure.com -P 3306 -p
```

2. At the `mysql` prompt, create a database.

```
CREATE DATABASE sampledb;
```

3. Create a database user called `phpappuser` and give it all privileges in the `sampledb` database. For simplicity of the tutorial, use `MySQLAzure2017` as the password.

```
CREATE USER 'phpappuser' IDENTIFIED BY 'MySQLAzure2017';
GRANT ALL PRIVILEGES ON sampledb.* TO 'phpappuser';
```

4. Exit the server connection by typing `quit`.

```
quit
```

## Connect app to Azure MySQL

In this step, you connect the PHP application to the MySQL database you created in Azure Database for MySQL.

### Configure the database connection

In the repository root, create an `.env.production` file and copy the following variables into it. Replace the placeholder \_<mysql-server-name>\_ in both `DB_HOST` and `DB_USERNAME`.

```
APP_ENV=production
APP_DEBUG=true
APP_KEY=

DB_CONNECTION=mysql
DB_HOST=<mysql-server-name>.mysql.database.azure.com
DB_DATABASE=sampledb
DB_USERNAME=phpappuser@<mysql-server-name>
DB_PASSWORD=MySQLAzure2017
MYSQL_SSL=true
```

Save the changes.

#### TIP

To secure your MySQL connection information, this file is already excluded from the Git repository (See `.gitignore` in the repository root). Later, you learn how to configure environment variables in App Service to connect to your database in Azure Database for MySQL. With environment variables, you don't need the `.env` file in App Service.

### Configure TLS/SSL certificate

By default, Azure Database for MySQL enforces TLS connections from clients. To connect to your MySQL database in Azure, you must use the [.pem certificate supplied by Azure Database for MySQL](#).

Open `config/database.php` and add the `sslmode` and `options` parameters to `connections.mysql`, as shown in the following code.

```
'mysql' => [
 ...
 'sslmode' => env('DB_SSLMODE', 'prefer'),
 'options' => (env('MYSQL_SSL')) ? [
 PDO::MYSQL_ATTR_SSL_KEY => '/ssl/BaltimoreCyberTrustRoot.crt.pem',
] : []
],
```

```
'mysql' => [
 ...
 'sslmode' => env('DB_SSLMODE', 'prefer'),
 'options' => (env('MYSQL_SSL') && extension_loaded('pdo_mysql')) ? [
 PDO::MYSQL_ATTR_SSL_KEY => '/ssl/BaltimoreCyberTrustRoot.crt.pem',
] : []
],
```

The certificate `BaltimoreCyberTrustRoot.crt.pem` is provided in the repository for convenience in this tutorial.

### Test the application locally

1. Run Laravel database migrations with `.env.production` as the environment file to create the tables in your MySQL database in Azure Database for MySQL. Remember that `.env.production` has the connection information to your MySQL database in Azure.

```
php artisan migrate --env=production --force
```

2. `.env.production` doesn't have a valid application key yet. Generate a new one for it in the terminal.

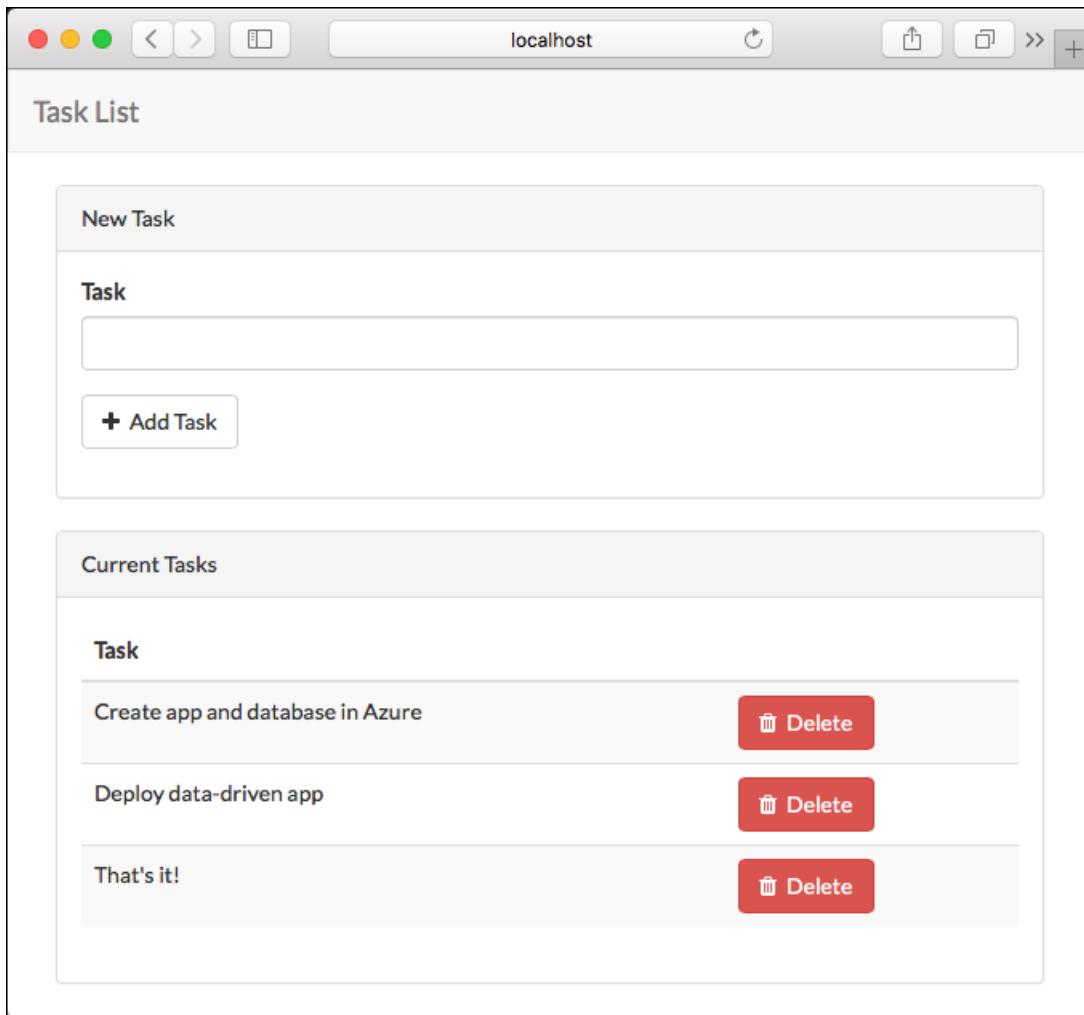
```
php artisan key:generate --env=production --force
```

3. Run the sample application with `.env.production` as the environment file.

```
php artisan serve --env=production
```

4. Navigate to `http://localhost:8000`. If the page loads without errors, the PHP application is connecting to the MySQL database in Azure.

5. Add a few tasks in the page.



6. To stop PHP, type `Ctrl + C` in the terminal.

### Commit your changes

Run the following Git commands to commit your changes:

```
git add .
git commit -m "database.php updates"
```

Your app is ready to be deployed.

## Deploy to Azure

In this step, you deploy the MySQL-connected PHP application to Azure App Service.

### Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the [az webapp deployment user set](#) command in Azure Cloud Shell. Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

## Create an App Service plan

In the Cloud Shell, create an App Service plan with the [az appservice plan create](#) command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "adminSiteName": null,
 "appServicePlanName": "myAppServicePlan",
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "app",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

In the Cloud Shell, create an App Service plan with the [az appservice plan create](#) command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "freeOfferExpirationTime": null,
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "linux",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

## Create a web app

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `PHP|7.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-
name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

You've created an empty new web app, with git deployment enabled.

### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

`https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `PHP|7.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.2" --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-
name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

You've created an empty new web app, with git deployment enabled.

#### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format `https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

## Configure database settings

In App Service, you set environment variables as *app settings* by using the `az webapp config appsettings set` command.

The following command configures the app settings `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`. Replace the placeholders `<app-name>` and `<mysql-server-name>`.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings DB_HOST="
<mysql-server-name>.mysql.database.azure.com" DB_DATABASE="sampledb" DB_USERNAME="phpappuser@<mysql-server-
name>" DB_PASSWORD="MySQLAzure2017" MYSQL_SSL="true"
```

You can use the PHP `getenv` method to access the settings. the Laravel code uses an `env` wrapper over the PHP `getenv`. For example, the MySQL configuration in `config/database.php` looks like the following code:

```
'mysql' => [
 'driver' => 'mysql',
 'host' => env('DB_HOST', 'localhost'),
 'database' => env('DB_DATABASE', 'forge'),
 'username' => env('DB_USERNAME', 'forge'),
 'password' => env('DB_PASSWORD', ''),
 ...
],
```

## Configure Laravel environment variables

Laravel needs an application key in App Service. You can configure it with app settings.

1. In the local terminal window, use `php artisan` to generate a new application key without saving it to `.env`.

```
php artisan key:generate --show
```

2. In the Cloud Shell, set the application key in the App Service app by using the

```
az webapp config appsettings set
```

command. Replace the placeholders `<app-name>` and `<outputofphpartisankey:generate>`.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings APP_KEY=<output_of_php_artisan_key:generate> APP_DEBUG="true"
```

`APP_DEBUG="true"` tells Laravel to return debugging information when the deployed app encounters errors. When running a production application, set it to `false`, which is more secure.

## Set the virtual application path

Set the virtual application path for the app. This step is required because the [Laravel application lifecycle](#) begins in the `public` directory instead of the application's root directory. Other PHP frameworks whose lifecycle start in the root directory can work without manual configuration of the virtual application path.

In the Cloud Shell, set the virtual application path by using the `az resource update` command. Replace the `<app-name>` placeholder.

```
az resource update --name web --resource-group myResourceGroup --namespace Microsoft.Web --resource-type config --parent sites/<app_name> --set properties.virtualApplications[0].physicalPath="site\wwwroot\public" --api-version 2015-06-01
```

By default, Azure App Service points the root virtual application path (`/`) to the root directory of the deployed application files (`sites\wwwroot`).

[Laravel application lifecycle](#) begins in the `public` directory instead of the application's root directory. The default PHP Docker image for App Service uses Apache, and it doesn't let you customize the `DocumentRoot` for Laravel. However, you can use `.htaccess` to rewrite all requests to point to `/public` instead of the root directory. In the repository root, an `.htaccess` is added already for this purpose. With it, your Laravel application is ready to be deployed.

For more information, see [Change site root](#).

## Push to Azure from Git

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace `<deploymentLocalGitUrl-from-create-step>` with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in [Configure a deployment user](#), not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

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

```
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 291 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id 'a5e076db9c'.
remote: Running custom deployment command...
remote: Running deployment command...
...
< Output has been truncated for readability >
```

#### NOTE

You may notice that the deployment process installs [Composer](#) packages at the end. App Service does not run these automations during default deployment, so this sample repository has three additional files in its root directory to enable it:

- `.deployment` - This file tells App Service to run `bash deploy.sh` as the custom deployment script.
- `deploy.sh` - The custom deployment script. If you review the file, you will see that it runs `php composer.phar install` after `npm install`.
- `composer.phar` - The Composer package manager.

You can use this approach to add any step to your Git-based deployment to App Service. For more information, see [Custom Deployment Script](#).

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace <deploymentLocalGitUrl-from-create-step> with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in [Configure a deployment user](#), not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

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

```
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 291 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id 'a5e076db9c'.
remote: Running custom deployment command...
remote: Running deployment command...
...
< Output has been truncated for readability >
```

## Browse to the Azure app

Browse to <http://<app-name>.azurewebsites.net> and add a few tasks to the list.

Congratulations, you're running a data-driven PHP app in Azure App Service.

## Update model locally and redeploy

In this step, you make a simple change to the `task` data model and the webapp, and then publish the update to Azure.

For the tasks scenario, you modify the application so that you can mark a task as complete.

### Add a column

1. In the local terminal window, navigate to the root of the Git repository.
2. Generate a new database migration for the `tasks` table:

```
php artisan make:migration add_complete_column --table=tasks
```

3. This command shows you the name of the migration file that's generated. Find this file in `database/migrations` and open it.
4. Replace the `up` method with the following code:

```
public function up()
{
 Schema::table('tasks', function (Blueprint $table) {
 $table->boolean('complete')->default(False);
 });
}
```

The preceding code adds a boolean column in the `tasks` table called `complete`.

- Replace the `down` method with the following code for the rollback action:

```
public function down()
{
 Schema::table('tasks', function (Blueprint $table) {
 $table->dropColumn('complete');
 });
}
```

- In the local terminal window, run Laravel database migrations to make the change in the local database.

```
php artisan migrate
```

Based on the [Laravel naming convention](#), the model `Task` (see `app/Task.php`) maps to the `tasks` table by default.

## Update application logic

- Open the `routes/web.php` file. The application defines its routes and business logic here.

- At the end of the file, add a route with the following code:

```
/**
 * Toggle Task completeness
 */
Route::post('/task/{id}', function ($id) {
 error_log('INFO: post /task/'.$id);
 $task = Task::findOrFail($id);

 $task->complete = !$task->complete;
 $task->save();

 return redirect('/');
});
```

The preceding code makes a simple update to the data model by toggling the value of `complete`.

## Update the view

- Open the `resources/views/tasks.blade.php` file. Find the `<tr>` opening tag and replace it with:

```
<tr class="{{ $task->complete ? 'success' : 'active' }}" >
```

The preceding code changes the row color depending on whether the task is complete.

- In the next line, you have the following code:

```
<td class="table-text"><div>{{ $task->name }}</div></td>
```

Replace the entire line with the following code:

```
<td>
<form action="{{ url('task/'.$task->id) }}" method="POST">
{{ csrf_field() }}

<button type="submit" class="btn btn-xs">
 <i class="fa {{ $task->complete ? 'fa-check-square-o' : 'fa-square-o'}}"></i>
</button>
{{ $task->name }}
</form>
</td>
```

The preceding code adds the submit button that references the route that you defined earlier.

### Test the changes locally

1. In the local terminal window, run the development server from the root directory of the Git repository.

```
php artisan serve
```

2. To see the task status change, navigate to <http://localhost:8000> and select the checkbox.

The screenshot shows a web application titled "Task List" running on a local host. The interface is divided into two main sections: "New Task" and "Current Tasks".

**New Task:** Contains a text input field labeled "Task" and a button labeled "+ Add Task".

**Current Tasks:** Displays a list of tasks with checkboxes and delete buttons:

- Create app and database in Azure (checkbox checked, Delete button)
- Deploy data-driven app (checkbox checked, Delete button)
- That's it! (checkbox unchecked, Delete button)

3. To stop PHP, type `ctrl + c` in the terminal.

### Publish changes to Azure

1. In the local terminal window, run Laravel database migrations with the production connection string to make the change in the Azure database.

```
php artisan migrate --env=production --force
```

2. Commit all the changes in Git, and then push the code changes to Azure.

```
git add .
git commit -m "added complete checkbox"
git push azure main
```

3. Once the `git push` is complete, navigate to the Azure app and test the new functionality.

The screenshot shows a web application titled "Task List" running on "php-mysql-123.azurewebsites.net". The interface is divided into two main sections: "New Task" and "Current Tasks".

- New Task:** Contains a text input field labeled "Task" and a button labeled "+ Add Task".
- Current Tasks:** Contains a list of three tasks, each with a checkbox and a "Delete" button:
  - Create app and database in Azure
  - Deploy app to Azure
  - That's it!

If you added any tasks, they are retained in the database. Updates to the data schema leave existing data intact.

## Stream diagnostic logs

While the PHP application runs in Azure App Service, you can get the console logs piped to your terminal. That way, you can get the same diagnostic messages to help you debug application errors.

To start log streaming, use the `az webapp log tail` command in the Cloud Shell.

```
az webapp log tail --name <app_name> --resource-group myResourceGroup
```

Once log streaming has started, refresh the Azure app in the browser to get some web traffic. You can now see console logs piped to the terminal. If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type `Ctrl + C`.

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging
filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

#### NOTE

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

#### TIP

A PHP application can use the standard `error_log()` to output to the console. The sample application uses this approach in `app/Http/routes.php`.

As a web framework, [Laravel](#) uses [Monolog](#) as the logging provider. To see how to get Monolog to output messages to the console, see [PHP: How to use monolog to log to console \(php://out\)](#).

## Manage the Azure app

1. Go to the [Azure portal](#) to manage the app you created.
2. From the left menu, click **App Services**, and then click the name of your Azure app.

The screenshot shows the Azure App Services overview page. On the left, there's a vertical navigation bar with icons for different services: App Service (highlighted with a red box), Functions, Logic Apps, Container Registry, and App Configuration. The main area has a dark header with the title 'App Services'. Below the header, there are buttons for '+ Add', 'Columns', and 'Refresh'. A message says 'Subscriptions: All 2 selected'. There are two filter inputs: 'Filter by name...' and 'All subscriptions'. A table lists one item: '1 items'. The columns are NAME, STATUS, APP TYPE, APP SERVICE PLAN, LOCATION, and SUBSCRI...'. The first row shows a blue icon, the name 'php-mysql-123' (also highlighted with a red box), 'Running', 'Web app', 'myAppServicePlan', 'West Europe', and 'Visual Studio ... \*\*\*'.

You see your app's Overview page. Here, you can perform basic management tasks like stop, start, restart, browse, and delete.

The left menu provides pages for configuring your app.

The screenshot shows the Azure portal's App Service overview for a resource named "php-mysql-123". The left sidebar includes links for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart, Deployment credentials, Deployment slots, Deployment options), and Help & support. The main content area displays the "Essentials" section with the following details:

- Resource group: myResourceGroup
- Status: Running
- Location: West Europe
- Subscription name: Visual Studio Ultimate with MSDN
- URL: http://php-mysql-123.azurewebsites.net

Below the essentials section is a "Monitoring" section titled "Requests and errors" which lists 4 items.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

In this tutorial, you learned how to:

- Create a MySQL database in Azure
- Connect a PHP app to MySQL
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to map a custom DNS name to the app.

[Tutorial: Map custom DNS name to your app](#)

Or, check out other resources:

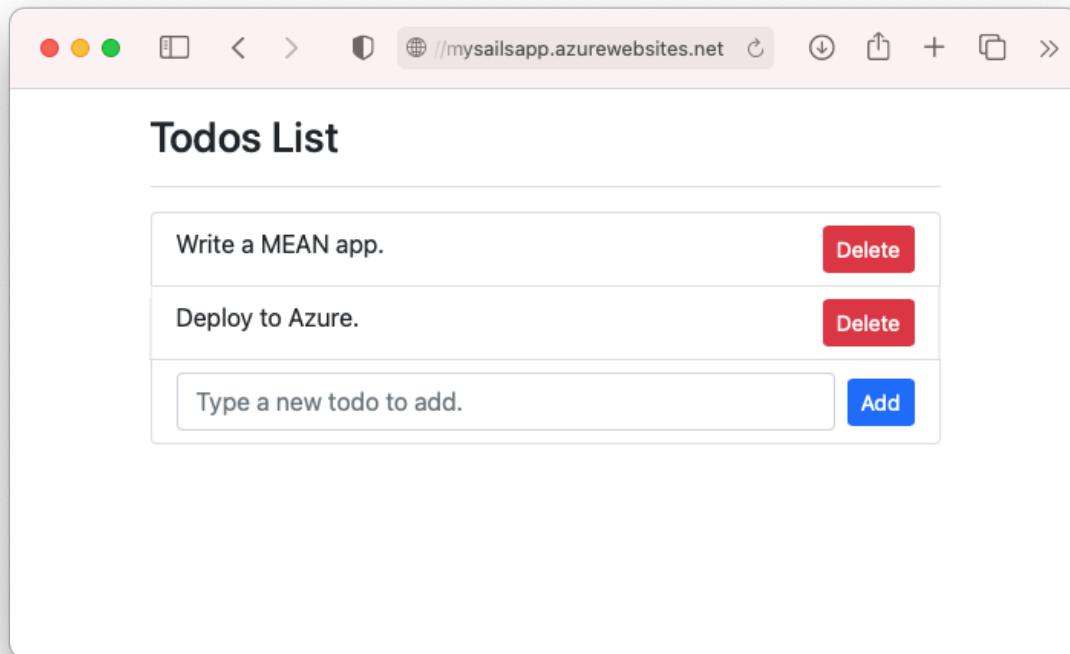
[Configure PHP app](#)

# Tutorial: Build a Node.js and MongoDB app in Azure

11/2/2021 • 17 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows how to create a Node.js app in App Service on Windows and connect it to a MongoDB database. When you're done, you'll have a MEAN application (MongoDB, Express, AngularJS, and Node.js) running in Azure App Service. The sample application uses a combination of [Sails.js](#) and [Angular 12](#).

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create a Node.js app in App Service on Linux, connect it locally to a MongoDB database, then deploy it to a database in Azure Cosmos DB's API for MongoDB. When you're done, you'll have a MEAN application (MongoDB, Express, AngularJS, and Node.js) running in App Service on Linux. The sample application uses a combination of [Sails.js](#) and [Angular 12](#).



What you'll learn:

- Create a MongoDB database in Azure
- Connect a Node.js app to MongoDB
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install Node.js and NPM](#)
- Use the Bash environment in [Azure Cloud Shell](#).



[Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Create local Node.js app

In this step, you set up the local Node.js project.

### Clone the sample application

In the terminal window, `cd` to a working directory.

Run the following command to clone the sample repository.

```
git clone https://github.com/Azure-Samples/mean-todoapp.git
```

#### NOTE

For information on how the sample app is created, see <https://github.com/Azure-Samples/mean-todoapp>.

### Run the application

Run the following commands to install the required packages and start the application.

```
cd mean-todoapp
npm install
node app.js --alter
```

When the app is fully loaded, you see something similar to the following message:

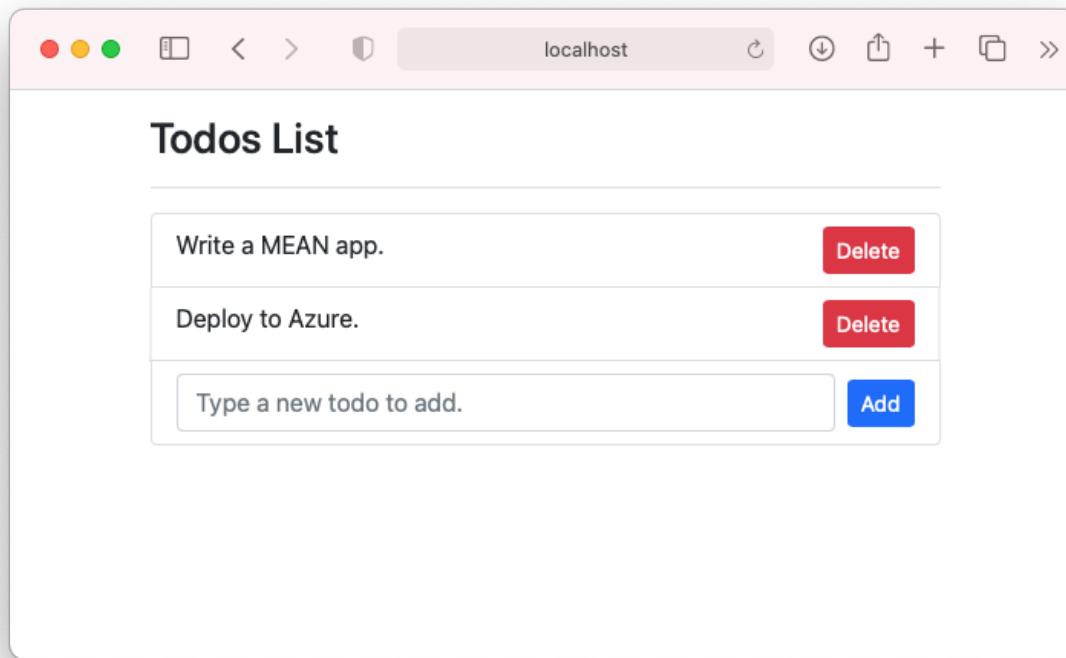
```
debug: -----
debug: :: Fri Jul 09 2021 13:10:34 GMT+0200 (Central European Summer Time)

debug: Environment : development
debug: Port : 1337
debug: -----
```

Navigate to `http://localhost:1337` in a browser. Add a few todo items.

The MEAN sample application stores user data in the database. By default, it uses a disk-based development

database. If you can create and see todo items, then your app is reading and writing data.



To stop Node.js at any time, press `ctrl+C` in the terminal.

## Create production MongoDB

In this step, you create a MongoDB database in Azure. When your app is deployed to Azure, it uses this cloud database.

For MongoDB, this tutorial uses [Azure Cosmos DB](#). Cosmos DB supports MongoDB client connections.

### Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named *myResourceGroup* in the *West Europe* location. To see all supported locations for App Service in **Free** tier, run the `az appservice list-locations --sku FREE` command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

### Create a Cosmos DB account

## NOTE

There is a cost to creating the Azure Cosmos DB databases in this tutorial in your own Azure subscription. To use a free Azure Cosmos DB account for seven days, you can use the [Try Azure Cosmos DB for free](#) experience. Just click the **Create** button in the MongoDB tile to create a free MongoDB database on Azure. Once the database is created, navigate to **Connection String** in the portal and retrieve your Azure Cosmos DB connection string for use later in the tutorial.

In the Cloud Shell, create a Cosmos DB account with the `az cosmosdb create` command.

In the following command, substitute a unique Cosmos DB name for the `<cosmosdb-name>` placeholder. This name is used as the part of the Cosmos DB endpoint, `https://<cosmosdb-name>.documents.azure.com/`, so the name needs to be unique across all Cosmos DB accounts in Azure. The name must contain only lowercase letters, numbers, and the hyphen (-) character, and must be between 3 and 50 characters long.

```
az cosmosdb create --name <cosmosdb-name> --resource-group myResourceGroup --kind MongoDB
```

The `--kind MongoDB` parameter enables MongoDB client connections.

When the Cosmos DB account is created, the Azure CLI shows information similar to the following example:

```
{
 "apiProperties": {
 "serverVersion": "3.6"
 },
 "backupPolicy": {
 "periodicModeProperties": {
 "backupIntervalInMinutes": 240,
 "backupRetentionIntervalInHours": 8,
 "backupStorageRedundancy": "Geo"
 },
 "type": "Periodic"
 },
 "capabilities": [
 {
 "name": "EnableMongo"
 }
],
 "connectorOffer": null,
 "consistencyPolicy": {
 "defaultConsistencyLevel": "Session",
 "maxIntervalInSeconds": 5,
 "maxStalenessPrefix": 100
 },
 "cors": [],
 "databaseAccountOfferType": "Standard",
 "defaultIdentity": "FirstPartyIdentity",
 "disableKeyBasedMetadataWriteAccess": false,
 "documentEndpoint": "https://<cosmosdb-name>.documents.azure.com:443/",
 ...
 < Output truncated for readability >
}
```

## Connect app to production MongoDB

In this step, you connect your sample application to the Cosmos DB database you just created, using a MongoDB connection string.

## Retrieve the database key

To connect to the Cosmos DB database, you need the database key. In the Cloud Shell, use the

```
az cosmosdb keys list
```

```
az cosmosdb keys list --name <cosmosdb-name> --resource-group myResourceGroup
```

The Azure CLI shows information similar to the following example:

```
{
 "primaryMasterKey": "RS4CmUwzGRASJPMoc0kiEvdnKmxyRILC9BWisAYh3Hq4zBYKr0XQiSE4pqx3UchBe04QRCzUt1i7w0r0kitoJw==",
 "primaryReadonlyMasterKey": "HvitsjIYz8TwRmIuPEUAALRwqgKOzJUjW22wPL2U8zoMvhGvregBkBk9LdMTxqBgDETSq7obbwZtdeFY7hElTg==",
 "secondaryMasterKey": "Lu9aeZTiXU4PjuuyGBbvS1N9IRG3oegIrIh95U6V0stf9bJiiIpw3IfwSUgQWSEYM3VeEyrhHJ4rn3Ci0vuFqA==",
 "secondaryReadonlyMasterKey": "LpsCicpVZqHRY7qbMgrzbRKjbYCwCKPQRl0QpgReAOxMcggTvxJFA94fTi0oQ7xtxpftTJcXkjTirQ0pT7QFrQ=="
}
```

Copy the value of `primaryMasterKey`. You need this information in the next step.

## Configure the connection string in your sample application

In your local repository, in `config/datastores.js`, replace the existing content with the following code and save your changes.

```
module.exports.datastores = {
 default: {
 adapter: 'sails-mongo',
 url: process.env.MONGODB_URI,
 ssl: true,
 },
};
```

The `ssl: true` option is required because [Cosmos DB requires TLS/SSL](#). `url` is set to an environment variable, which you will set next.

In the terminal, set the `MONGODB_URI` environment variable. Be sure to replace the two `<cosmosdb-name>` placeholders with your Cosmos DB database name, and replace the `<cosmosdb-key>` placeholder with the key you copied in the previous step.

```
export MONGODB_URI=mongodb://<cosmosdb-name>:<cosmosdb-key>@<cosmosdb-name>.documents.azure.com:10250/todoapp
```

### NOTE

This connection string follows the format defined in the [Sails.js documentation](#).

## Test the application with MongoDB

In a local terminal window, run `node app.js --alter` again.

```
node app.js --alter
```

Navigate to <http://localhost:1337> again. If you can create and see todo items, then your app is reading and writing data using the Cosmos DB database in Azure.

In the terminal, stop Node.js by typing `Ctrl+C`.

## Deploy app to Azure

In this step, you deploy your MongoDB-connected Node.js application to Azure App Service.

### Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the `az webapp deployment user set` command in Azure Cloud Shell.

Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

### Create an App Service plan

In the Cloud Shell, create an App Service plan with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **B1** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku B1
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "freeOfferExpirationTime": null,
 "geoRegion": "UK West",
 "hostingEnvironmentProfile": null,
 "hyperV": false,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "isSpot": false,
 "isXenon": false,
 "kind": "app",
 "location": "ukwest",
 "maximumElasticWorkerCount": 1,
 "maximumNumberOfWorkers": 0,
 < JSON data removed for brevity. >
}
```

In the Cloud Shell, create an App Service plan with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **B1** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku B1 --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "freeOfferExpirationTime": null,
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "linux",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

## Create a web app

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `NODE|14-lts`. To see all supported runtimes, run `az webapp list-runtimes`.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
'NODE|14-lts' --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-
name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

You've created an empty web app, with git deployment enabled.

### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

```
https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git
```

Save this URL as you need it later.

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `NODE|14-lts`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime 'NODE|14-lts' --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "clientCertExclusionPaths": null,
 "clientCertMode": "Required",
 "cloningInfo": null,
 "containerSize": 0,
 "customDomainVerificationId": "54184270DF7B3B4BF30221B6303E789C324E4783C8DF1FBAA3D111FC72328CE9",
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

You've created an empty web app, with git deployment enabled.

#### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

```
https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git
```

### Configure an environment variable

Remember that the sample application is already configured to use the `MONGODB_URI` environment variable in `config/datastores.js`. In App Service, you inject this variable by using an [app setting](#).

To set app settings, use the `az webapp config appsettings set` command in the Cloud Shell.

The following example configures a `MONGODB_URI` app setting in your Azure app. Replace the `<app-name>`, `<cosmosdb-name>`, and `<cosmosdb-key>` placeholders.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
MONGODB_URI='mongodb://<cosmosdb-name>:<cosmosdb-key>@<cosmosdb-name>.documents.azure.com:10250/todoapp'
DEPLOYMENT_BRANCH='main'
```

#### NOTE

`DEPLOYMENT_BRANCH` is a special app setting that tells the deployment engine which Git branch you're deploying to in App Service.

## Push to Azure from Git

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace `<deploymentLocalGitUrl-from-create-step>` with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in [Configure a deployment user](#), not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

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

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 318 bytes | 318.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id '4eb0ca7190'.
remote: Generating deployment script.
remote: Running deployment command...
remote: Handling node.js deployment.
remote: Creating app_offline.htm
remote: KuduSync.NET from: 'D:\home\site\repository' to: 'D:\home\site\wwwroot'
remote: Copying file: 'package.json'
remote: Deleting app_offline.htm
remote: Looking for app.js/server.js under site root.
remote: Using start-up script app.js
remote: Generated web.config.
.
.
.
remote: Deployment successful.
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
 * [new branch] main -> main
```

**TIP**

During Git deployment, the deployment engine runs `npm install --production` as part of its build automation.

- As defined in `package.json`, the `postinstall` script is picked up by `npm install` and runs `ng build` to generate the production files for Angular and deploy them to the `assets` folder.
- `scripts` in `package.json` can use tools that are installed in `node_modules/.bin`. Since `npm install` has installed `node_modules/.bin/ng` too, you can use it to deploy your Angular client files. This npm behavior is exactly the same in Azure App Service. Packages under `devDependencies` in `package.json` are not installed. Any package you need in the production environment needs to be moved under `dependencies`.

If your app needs to bypass the default automation and run custom automation, see [Run Grunt/Bower/Gulp](#).

```
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 347 bytes | 347.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Deploy Async
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id 'f776be774a'.
remote: Repository path is /home/site/repository
remote: Running oryx build...
remote: Operation performed by Microsoft Oryx, https://github.com/Microsoft/Oryx
remote: You can report issues at https://github.com/Microsoft/Oryx/issues
remote:
remote: Oryx Version: 0.2.20210420.1, Commit: 85c6e9278aae3980b86cb1d520aaad532c814ed7,
ReleaseTagName: 20210420.1
remote:
remote: Build Operation ID: |qwejn9R4StI=.5e8a3529_
remote: Repository Commit : f776be774a3ea8abc48e5ee2b5132c037a636f73
.
.
.
remote: Deployment successful.
remote: Deployment Logs : 'https://<app-name>.scm.azurewebsites.net/newui/jsonviewer?view_url=/api/deployments/a6fcf811136739f145e0de3be82ff195bca7a68b/log'
To https://<app-name>.scm.azurewebsites.net/<app-name>.git
 4f7e3ac..a6fcf81 main -> main
```

**TIP**

During Git deployment, the deployment engine runs `npm install` as part of its build automation.

- As defined in `package.json`, the `postinstall` script is picked up by `npm install` and runs `ng build` to generate the production files for Angular and deploy them to the `assets` folder.
- `scripts` in `package.json` can use tools that are installed in `node_modules/.bin`. Since `npm install` has installed `node_modules/.bin/ng` too, you can use it to deploy your Angular client files. This npm behavior is exactly the same in Azure App Service. When build automation is complete, the whole completed repository is copied into the `/home/site/wwwroot` folder, out of which your app is hosted.

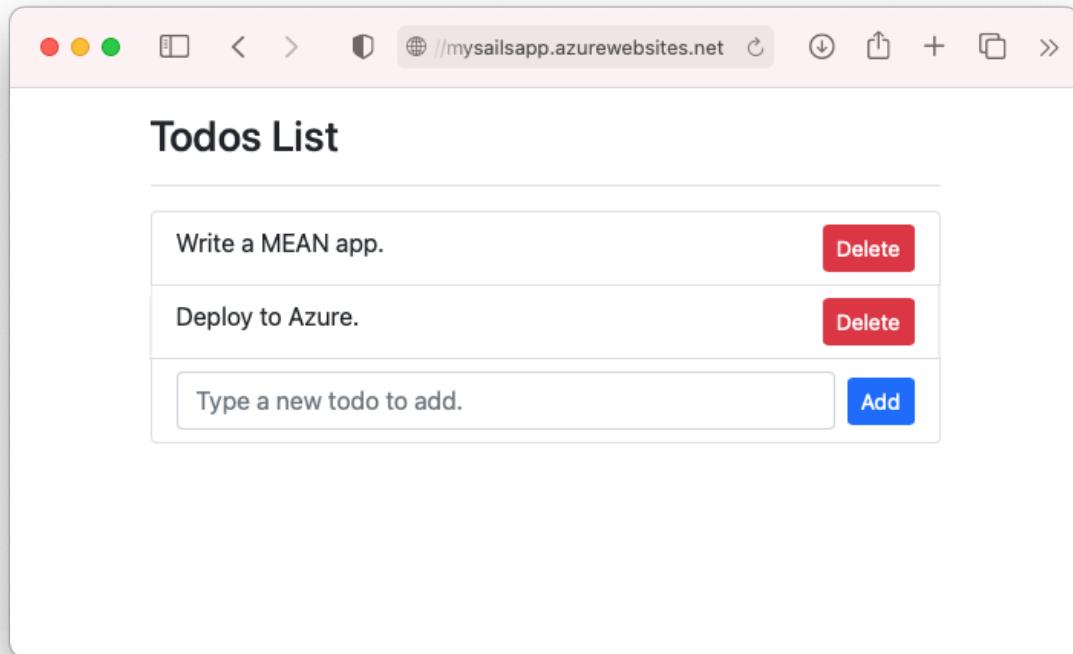
If your app needs to bypass the default automation and run custom automation, see [Run Grunt/Bower/Gulp](#).

## Browse to the Azure app

Browse to the deployed app using your web browser.

<https://<app-name>.azurewebsites.net>

If you can create and see todo items in the browser, then your sample app in Azure has connectivity to the MongoDB (Cosmos DB) database.



**Congratulations!** You're running a data-driven Node.js app in Azure App Service.

## Update data model and redeploy

In this step, you change the `Todo` data model and publish your change to Azure.

### Update the server-side model

In Sails.js, changing the server-side model and API code is as simple as changing the data model, because [Sails.js already defines the common routes](#) for a model by default.

In your local repository, open `api/models/Todo.js` and add a `done` attribute. When you're done, your schema code should look like this:

```
module.exports = {

 attributes: {
 value: {type: 'string'},
 done: {type: 'boolean', defaultsTo: false}
 },

};
```

### Update the client code

There are three files you need to modify: the client model, the HTML template, and the component file.

Open `client/src/app/todo.ts` and add a `done` property. When you're done, your model show look like this:

```
export class Todo {
 id!: String;
 value!: String;
 done!: Boolean;
}
```

Open `client/src/app/app.component.html`. Just above the only `<span>` element, add the following code to add a checkbox at the beginning of each todo item:

```
<input class="form-check-input me-2" type="checkbox" [checked]="todo.done" (click)="toggleDone(todo.id, i)"
[disabled]="isProcessing">
```

Open `client/src/app/app.component.ts`. Just above the last closing curly brace (`}`), insert the following method. It's called by the template code above when the checkbox is clicked and updates the server-side data.

```
toggleDone(id:any, i:any) {
 console.log("Toggled checkbox for " + id);
 this.isProcessing = true;
 this.Todos[i].done = !this.Todos[i].done;
 this.restService.updateTodo(id, this.Todos[i])
 .subscribe((res) => {
 console.log('Data updated successfully!');
 this.isProcessing = false;
 }, (err) => {
 console.log(err);
 this.Todos[i].done = !this.Todos[i].done;
 });
}
```

## Test your changes locally

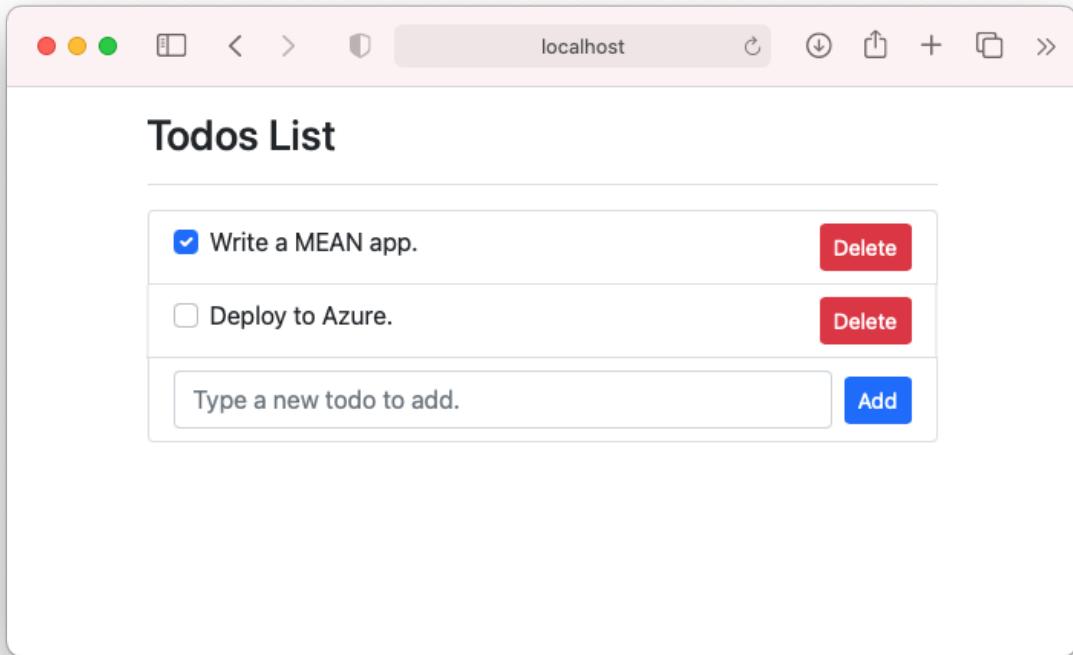
In the local terminal window, compile the updated Angular client code with the build script defined in `package.json`.

```
npm run build
```

Test your changes with `node app.js --alter` again. Since you changed your server-side model, the `--alter` flag lets `sails.js` alter the data structure in your Cosmos DB database.

```
node app.js --alter
```

Navigate to `http://localhost:1337`. You should now see a checkbox in front of todo item. When you select or clear a checkbox, the Cosmos DB database in Azure is updated to indicate that the todo item is done.



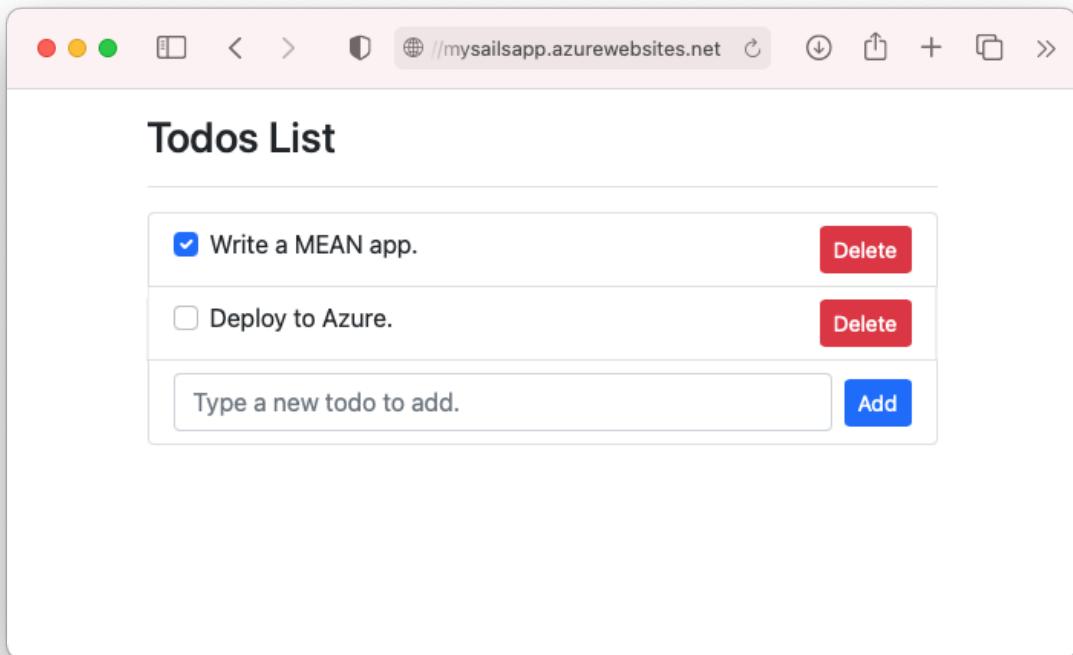
In the terminal, stop Nodejs by typing `Ctrl+C`.

### Publish changes to Azure

In the local terminal window, commit your changes in Git, then push the code changes to Azure.

```
git commit -am "added done field"
git push azure main
```

Once the `git push` is complete, navigate to your Azure app and try out the new functionality.



If you added any articles earlier, you still can see them. Existing data in your Cosmos DB is not lost. Also, your updates to the data schema and leaves your existing data intact.

## Stream diagnostic logs

While your Node.js application runs in Azure App Service, you can get the console logs piped to your terminal. That way, you can get the same diagnostic messages to help you debug application errors.

To start log streaming, use the `az webapp log tail` command in the Cloud Shell.

```
az webapp log tail --name <app-name> --resource-group myResourceGroup
```

Once log streaming has started, refresh your Azure app in the browser to get some web traffic. You now see console logs piped to your terminal.

Stop log streaming at any time by typing `Ctrl+C`.

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

### NOTE

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

## Manage your Azure app

Go to the [Azure portal](#) to see the app you created.

From the left menu, click **App Services**, then click the name of your Azure app.

The screenshot shows the Azure App Services portal. On the left, there's a sidebar with icons for Add, Columns, Refresh, Subscriptions, Filter by name..., All subscriptions, 1 items, NAME, STATUS, APP TYPE, APP SERVICE PLAN, and LOCATION. Below this is a table with one row. The first column has a globe icon and the name 'meanjs'. This entire first column is highlighted with a red box. The other columns show 'Running', 'Web app', 'myAppServicePlan', and 'West Europe'. The 'NAME' column header is also highlighted with a red box.

By default, the portal shows your app's **Overview** page. This page gives you a view of how your app is doing. Here, you can also perform basic management tasks like browse, stop, start, restart, and delete. The tabs on the left side of the page show the different configuration pages you can open.

The screenshot shows the Azure App Service Overview page for the 'meanjs' app. The left sidebar includes tabs for Overview (which is selected and highlighted in blue), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Quickstart, and Deployment credentials. The main content area has tabs for Browse, Stop, Swap, Restart, Delete, and More. Under the Essentials section, it shows the Resource group (myResourceGroup), Status (Running), Location (West Europe), Subscription name (myAppServicePlan (Free)), Subscription ID, URL (http://meanjs.azurewebsites.net), App Service plan/pricing tier (myAppServicePlan (Free)), Git/Deployment username, Git clone url, and FTP hostname (ftp://waws-prod-am2-025.ftp.azurewebsites.net). Under the Monitoring section, there's a Requests and errors tab.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

What you learned:

- Create a MongoDB database in Azure
- Connect a Node.js app to MongoDB
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream logs from Azure to your terminal
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to map a custom DNS name to the app.

[Map an existing custom DNS name to Azure App Service](#)

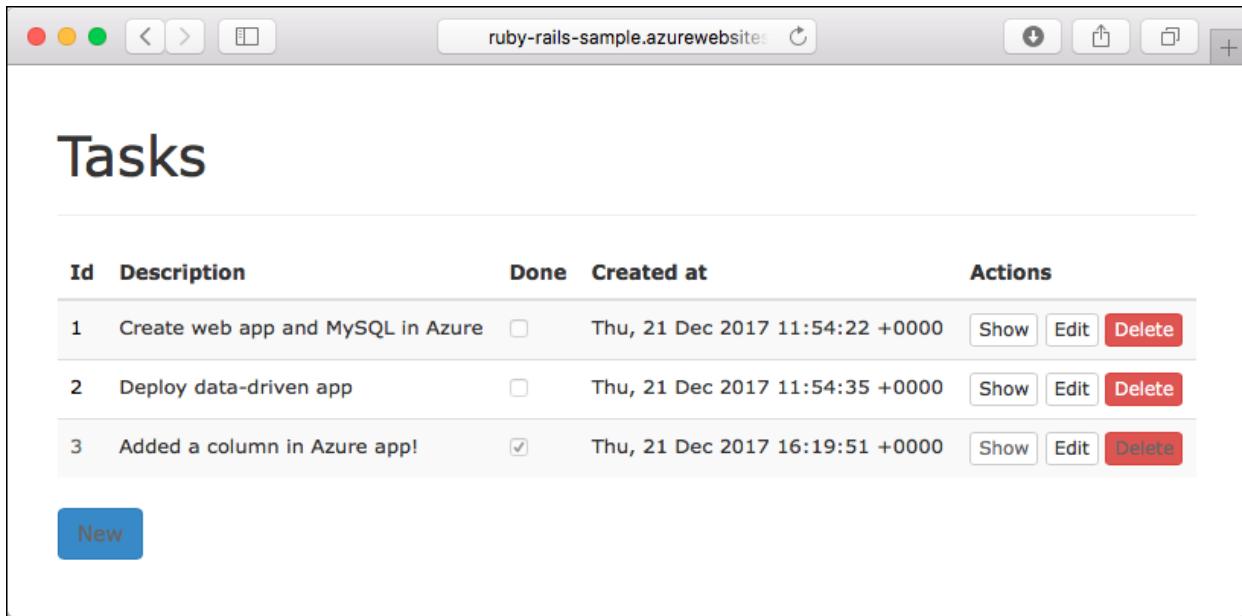
Or, check out other resources:

- [Configure Node.js app](#)
- [Environment variables and app settings reference](#)

# Build a Ruby and Postgres app in Azure App Service on Linux

11/2/2021 • 13 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows how to create a Ruby app and connect it to a PostgreSQL database. When you're finished, you'll have a [Ruby on Rails](#) app running on App Service on Linux.



The screenshot shows a browser window with the URL `ruby-rails-sample.azurewebsites`. The page title is "Tasks". Below the title is a table with three rows of tasks:

ID	Description	Done	Created at	Actions
1	Create web app and MySQL in Azure	<input type="checkbox"/>	Thu, 21 Dec 2017 11:54:22 +0000	Show Edit Delete
2	Deploy data-driven app	<input type="checkbox"/>	Thu, 21 Dec 2017 11:54:35 +0000	Show Edit Delete
3	Added a column in Azure app!	<input checked="" type="checkbox"/>	Thu, 21 Dec 2017 16:19:51 +0000	Show Edit Delete

At the bottom left of the table is a blue "New" button.

In this tutorial, you learn how to:

- Create a PostgreSQL database in Azure
- Connect a Ruby on Rails app to PostgreSQL
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install Ruby 2.6](#)
- [Install Ruby on Rails 5.1](#)
- [Install and run PostgreSQL](#)
- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.

- If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
- When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
- Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

## Prepare local Postgres

In this step, you create a database in your local Postgres server for your use in this tutorial.

### Connect to local Postgres server

1. Open the terminal window and run `psql` to connect to your local Postgres server.

```
sudo -u postgres psql
```

If your connection is successful, your Postgres database is running. If not, make sure that your local Postgres database is started by following the steps at [Downloads - PostgreSQL Core Distribution](#).

2. Type `\q` to exit the Postgres client.
3. Create a Postgres user that can create databases by running the following command, using your signed-in Linux username.

```
sudo -u postgres createuser -d <signed-in-user>
```

## Create a Ruby on Rails app locally

In this step, you get a Ruby on Rails sample application, configure its database connection, and run it locally.

### Clone the sample

1. In the terminal window, `cd` to a working directory.
2. Clone the sample repository and change to the repository root.

```
git clone https://github.com/Azure-Samples/rubyrails-tasks.git
cd rubyrails-tasks
```

3. Make sure the default branch is `main`.

```
git branch -m main
```

#### TIP

The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main`, this tutorial also shows you how to deploy a repository from `main`. For more information, see [Change deployment branch](#).

4. Install the required packages.

```
bundle install --path vendor/bundle
```

## Run the sample locally

1. Run [the Rails migrations](#) to create the tables the application needs. To see which tables are created in the migrations, look in the `db/migrate` directory in the Git repository.

```
rake db:create
rake db:migrate
```

2. Run the application.

```
rails server
```

3. Navigate to <http://localhost:3000> in a browser. Add a few tasks in the page.

ID	Description	Created at	Actions
3	Create Linux web app and db in Azure	Thu, 21 Dec 2017 11:10:23 +0000	Show Edit Delete
4	Deploy data-driven app	Thu, 21 Dec 2017 11:10:34 +0000	Show Edit Delete

4. To stop the Rails server, type `ctrl + c` in the terminal.

## Create Postgres in Azure

In this step, you create a Postgres database in [Azure Database for PostgreSQL](#). Later, you configure the Ruby on Rails application to connect to this database.

### Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named `myResourceGroup` in the `West Europe` location. To see all supported locations for App Service on Linux in **Basic** tier, run the `az appservice list-locations --sku B1 --linux-workers-enabled` command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

# Create Postgres database in Azure

1. Install the `db-up` extension with the following command:

```
az extension add --name db-up
```

2. Create the Postgres database in Azure with the `az postgres up` command, as shown in the following example. Replace `<postgresql-name>` with a *unique name* (the server endpoint is `https://<postgresql-name>.postgres.database.azure.com`). For `<admin-username>` and `<admin-password>`, specify credentials to create an administrator user for this Postgres server.

```
az postgres up --resource-group myResourceGroup --location westeurope --server-name <postgresql-name>
--database-name sampledb --admin-user <admin-username> --admin-password <admin-password> --ssl-enforcement Enabled
```

This command may take a while because it's doing the following:

- Creates a [resource group](#) called `myResourceGroup`, if it doesn't exist. Every Azure resource needs to be in one of these. `--resource-group` is optional.
- Creates a Postgres server with the administrative user.
- Creates a `sampledb` database.
- Allows access from your local IP address.
- Allows access from Azure services.
- Create a database user with access to the `sampledb` database.

You can do all the steps separately with other `az postgres` commands and `psql`, but `az postgres up` does all of them in one step for you.

When the command finishes, find the output lines that begin with `Ran Database Query:`. They show the database user that's created for you, with the username `root` and password `Sampledb1`. You'll use them later to connect your app to the database.

## TIP

`--location <location-name>`, can be set to any one of the [Azure regions](#). You can get the regions available to your subscription with the `az account list-locations` command. For production apps, put your database and your app in the same location.

## Connect app to Azure Postgres

In this step, you connect the Ruby on Rails application to the Postgres database you created in Azure Database for PostgreSQL.

### Configure the database connection

In the repository, open `config/database.yml`. At the bottom of the file, replace the production variables with the following code.

```
production:
 <<: *default
 host: <%= ENV['DB_HOST'] %>
 database: <%= ENV['DB_DATABASE'] %>
 username: <%= ENV['DB_USERNAME'] %>
 password: <%= ENV['DB_PASSWORD'] %>
```

Save the changes.

## Test the application locally

1. Back in the local terminal, set the following environment variables:

```
export DB_HOST=<postgres-server-name>.postgres.database.azure.com
export DB_DATABASE=sampledbs
export DB_USERNAME=root@<postgres-server-name>
export DB_PASSWORD=Sampledb1
```

2. Run Rails database migrations with the production values you just configured to create the tables in your Postgres database in Azure Database for PostgreSQL.

```
rake db:migrate RAILS_ENV=production
```

3. When running in the production environment, the Rails application needs precompiled assets. Generate the required assets with the following command:

```
rake assets:precompile
```

4. The Rails production environment also uses secrets to manage security. Generate a secret key.

```
rails secret
```

5. Save the secret key to the respective variables used by the Rails production environment. For convenience, you use the same key for both variables.

```
export RAILS_MASTER_KEY=<output-of-rails-secret>
export SECRET_KEY_BASE=<output-of-rails-secret>
```

6. Enable the Rails production environment to serve JavaScript and CSS files.

```
export RAILS_SERVE_STATIC_FILES=true
```

7. Run the sample application in the production environment.

```
rails server -e production
```

8. Navigate to <http://localhost:3000>. If the page loads without errors, the Ruby on Rails application is connecting to the Postgres database in Azure.

9. Add a few tasks in the page.

ID	Description	Created at	Actions
1	Create web app and MySQL in Azure	Thu, 21 Dec 2017 11:54:22 +0000	Show Edit Delete
2	Deploy data-driven app	Thu, 21 Dec 2017 11:54:35 +0000	Show Edit Delete

New

10. To stop the Rails server, type `ctrl + c` in the terminal.

### Commit your changes

1. Run the following Git commands to commit your changes:

```
git add .
git commit -m "database.yml updates"
```

Your app is ready to be deployed.

## Deploy to Azure

In this step, you deploy the Postgres-connected Rails application to Azure App Service.

### Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the `az webapp deployment user set` command in Azure Cloud Shell.

Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

### Create an App Service plan

In the Cloud Shell, create an App Service plan with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE --is-linux
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "freeOfferExpirationTime": null,
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "linux",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

## Create a web app

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`). The runtime is set to `RUBY|2.6.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
'RUBY|2.6.2' --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-
name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

You've created an empty new web app, with git deployment enabled.

### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

```
https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git
```

## Configure database settings

In App Service, you set environment variables as *app settings* by using the `az webapp config appsettings set` command in the Cloud Shell.

The following Cloud Shell command configures the app settings `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`. Replace the placeholders `<appname>` and `<postgres-server-name>`.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings DB_HOST="<postgres-server-name>.postgres.database.azure.com" DB_DATABASE="sampledb" DB_USERNAME="root@<postgres-server-name>" DB_PASSWORD="Sampledb1"
```

## Configure Rails environment variables

1. In the local terminal, [generate a new secret](#) for the Rails production environment in Azure.

```
rails secret
```

2. In the following Cloud Shell command, replace the two `<output-of-rails-secret>` placeholders with the new secret key you generated in the local terminal.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
RAILS_MASTER_KEY="<output-of-rails-secret>" SECRET_KEY_BASE="<output-of-rails-secret>"
RAILS_SERVE_STATIC_FILES="true" ASSETS_PRECOMPILE="true"
```

`ASSETS_PRECOMPILE="true"` tells the default Ruby container to precompile assets at each Git deployment. For more information, see [Precompile assets](#) and [Serve static assets](#).

## Push to Azure from Git

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the [az webapp config appsettings set](#) command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
DEPLOYMENT_BRANCH='main'
```

2. In the local terminal, add an Azure remote to your local Git repository.

```
git remote add azure <paste-copied-url-here>
```

3. Push to the Azure remote to deploy the Ruby on Rails application. You are prompted for the password you supplied earlier as part of the creation of the deployment user.

```
git push azure main
```

During deployment, Azure App Service communicates its progress with Git.

```
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 291 bytes | 0 bytes/s, done.
Total 3 (delta 2), reused 0 (delta 0)
remote: Updating branch 'main'.
remote: Updating submodules.
remote: Preparing deployment for commit id 'a5e076db9c'.
remote: Running custom deployment command...
remote: Running deployment command...
...
< Output has been truncated for readability >
```

## Browse to the Azure app

Browse to <http://<app-name>.azurewebsites.net> and add a few tasks to the list.

ID	Description	Created at	Actions
1	Create web app and MySQL in Azure	Thu, 21 Dec 2017 11:54:22 +0000	Show Edit Delete
2	Deploy data-driven app	Thu, 21 Dec 2017 11:54:35 +0000	Show Edit Delete

Congratulations, you're running a data-driven Ruby on Rails app in Azure App Service.

## Update model locally and redeploy

In this step, you make a simple change to the `task` data model and the webapp, and then publish the update to Azure.

For the tasks scenario, you modify the application so that you can mark a task as complete.

### Add a column

1. In the terminal, navigate to the root of the Git repository.
2. Generate a new migration that adds a boolean column called `Done` to the `Tasks` table:

```
rails generate migration AddDoneToTasks Done:boolean
```

This command generates a new migration file in the `db/migrate` directory.

3. In the terminal, run Rails database migrations to make the change in the local database.

```
rake db:migrate
```

## Update application logic

1. Open the `app/controllers/tasks_controller.rb` file. At the end of the file, find the following line:

```
params.require(:task).permit(:Description)
```

2. Modify this line to include the new `:Done` parameter.

```
params.require(:task).permit(:Description, :Done)
```

## Update the views

1. Open the `app/views/tasks/_form.html.erb` file, which is the Edit form.

2. Find the line `<%= f.error_span(:Description) %>` and insert the following code directly below it:

```
<%= f.label :Done, :class => 'control-label col-lg-2' %>
<div class="col-lg-10">
 <%= f.check_box :Done, :class => 'form-control' %>
</div>
```

3. Open the `app/views/tasks/show.html.erb` file, which is the single-record View page.

Find the line `<dd><%= @task.Description %></dd>` and insert the following code directly below it:

```
<dt><%= model_class.human_attribute_name(:Done) %></dt>
<dd><%= check_box "task", "Done", {:checked => @task.Done, :disabled => true}></dd>
```

Open the `app/views/tasks/index.html.erb` file, which is the Index page for all records.

Find the line `<th><%= model_class.human_attribute_name(:Description) %></th>` and insert the following code directly below it:

```
<th><%= model_class.human_attribute_name(:Done) %></th>
```

4. In the same file, find the line `<td><%= task.Description %></td>` and insert the following code directly below it:

```
<td><%= check_box "task", "Done", {:checked => task.Done, :disabled => true}></td>
```

## Test the changes locally

1. In the local terminal, run the Rails server.

```
rails server
```

2. To see the task status change, navigate to `http://localhost:3000` and add or edit items.

Id	Description	Done	Created at	Actions
3	Create Linux web app and db in Azure	<input type="checkbox"/>	Thu, 21 Dec 2017 11:10:23 +0000	Show Edit Delete
4	Deploy data-driven app	<input type="checkbox"/>	Thu, 21 Dec 2017 11:10:34 +0000	Show Edit Delete
8	Added checkbox to the right!	<input checked="" type="checkbox"/>	Thu, 21 Dec 2017 16:04:25 +0000	Show Edit Delete

3. To stop the Rails server, type `Ctrl + C` in the terminal.

### Publish changes to Azure

1. In the terminal, run Rails database migrations for the production environment to make the change in the Azure database.

```
rake db:migrate RAILS_ENV=production
```

2. Commit all the changes in Git, and then push the code changes to Azure.

```
git add .
git commit -m "added complete checkbox"
git push azure main
```

3. Once the `git push` is complete, navigate to the Azure app and test the new functionality.

Id	Description	Done	Created at	Actions
1	Create web app and MySQL in Azure	<input type="checkbox"/>	Thu, 21 Dec 2017 11:54:22 +0000	Show Edit Delete
2	Deploy data-driven app	<input type="checkbox"/>	Thu, 21 Dec 2017 11:54:35 +0000	Show Edit Delete
3	Added a column in Azure app!	<input checked="" type="checkbox"/>	Thu, 21 Dec 2017 16:19:51 +0000	Show Edit Delete

If you added any tasks, they are retained in the database. Updates to the data schema leave existing data intact.

## Stream diagnostic logs

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

### NOTE

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

## Manage the Azure app

1. Go to the [Azure portal](#) to manage the app you created.
2. From the left menu, click **App Services**, and then click the name of your Azure app.

The screenshot shows the Azure portal's App Services overview. On the left, there's a sidebar with icons for different services. The main area has a header 'App Services'. Below it, there's a search bar and a dropdown for 'All subscriptions'. The table lists one item: 'php-mysql-123', which is a 'Web app' running in 'myAppServicePlan' located in 'West Europe'. The 'NAME' column is highlighted with a red box.

NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCRI...
php-mysql-123	Running	Web app	myAppServicePlan	West Europe	Visual Studio ...

You see your app's Overview page. Here, you can perform basic management tasks like stop, start, restart, browse, and delete.

The left menu provides pages for configuring your app.

The screenshot shows the Azure portal interface for managing an App Service. The left sidebar lists various management options. The main content area is divided into sections like 'Essentials' and 'Monitoring'. Under 'Essentials', detailed information about the resource group, status, location, and subscription is provided, along with deployment and monitoring URLs. The 'Monitoring' section includes a table for tracking requests and errors.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

In this tutorial, you learned how to:

- Create a Postgres database in Azure
- Connect a Ruby on Rails app to Postgres
- Deploy the app to Azure
- Update the data model and redeploy the app
- Stream diagnostic logs from Azure
- Manage the app in the Azure portal

Advance to the next tutorial to learn how to map a custom DNS name to your app.

[Tutorial: Map custom DNS name to your app](#)

Or, check out other resources:

[Configure Ruby app](#)

# Tutorial: Deploy a Django web app with PostgreSQL in Azure App Service

11/2/2021 • 17 minutes to read • [Edit Online](#)

This tutorial shows how to deploy a data-driven Python [Django](#) web app to [Azure App Service](#) and connect it to an Azure Database for Postgres database. You can also try the PostgreSQL Flexible Server (Preview) by selecting the option above. Flexible Server provides a simpler deployment mechanism and lower ongoing costs.

In this tutorial, you use the Azure CLI to complete the following tasks:

- Set up your initial environment with Python and the Azure CLI
- Create an Azure Database for PostgreSQL database
- Deploy code to Azure App Service and connect to PostgreSQL
- Update your code and redeploy
- View diagnostic logs
- Manage the web app in the Azure portal

You can also use the [Azure portal version of this tutorial](#).

This tutorial shows how to deploy a data-driven Python [Django](#) web app to [Azure App Service](#) and connect it to an [Azure Database for PostgreSQL Flexible Server \(Preview\)](#) database. If you cannot use PostgreSQL Flexible Server (Preview), then select the Single Server option above.

In this tutorial, you use the Azure CLI to complete the following tasks:

- Set up your initial environment with Python and the Azure CLI
- Create an Azure Database for PostgreSQL Flexible Server database
- Deploy code to Azure App Service and connect to PostgreSQL Flexible Server
- Update your code and redeploy
- View diagnostic logs
- Manage the web app in the Azure portal

You can also use the [Azure portal version of this tutorial](#).

## 1. Set up your initial environment

1. Have an Azure account with an active subscription. [Create an account for free](#).
2. Install [Python 3.6 or higher](#).
3. Install the [Azure CLI](#) 2.18.0 or higher, with which you run commands in any shell to provision and configure Azure resources.

Open a terminal window and check your Python version is 3.6 or higher:

- [Bash](#)
- [PowerShell](#)
- [Cmd](#)

```
python3 --version
```

Check that your Azure CLI version is 2.18.0 or higher:

```
az --version
```

If you need to upgrade, try the `az upgrade` command (requires version 2.11+) or see [Install the Azure CLI](#).

Then sign in to Azure through the CLI:

```
az login
```

This command opens a browser to gather your credentials. When the command finishes, it shows JSON output containing information about your subscriptions.

Once signed in, you can run Azure commands with the Azure CLI to work with resources in your subscription.

Having issues? [Let us know](#).

## 2. Clone or download the sample app

- [Git clone](#)
- [Download](#)

Clone the sample repository:

```
git clone https://github.com/Azure-Samples/djangoapp
```

Then navigate into that folder:

```
cd djangoapp
```

For Flexible Server (Preview), use the flexible-server branch of the sample, which contains a few necessary changes, such as how the database server URL is set and adding `'OPTIONS': {'sslmode': 'require'}` to the Django database configuration as required by Azure PostgreSQL Flexible Server.

```
git checkout flexible-server
```

The djangoapp sample contains the data-driven Django polls app you get by following [Writing your first Django app](#) in the Django documentation. The completed app is provided here for your convenience.

The sample is also modified to run in a production environment like App Service:

- Production settings are in the `azuresite/production.py` file. Development settings are in `azuresite/settings.py`.
- The app uses production settings when the `WEBSITE_HOSTNAME` environment variable is set. Azure App Service automatically sets this variable to the URL of the web app, such as `msdocs-django.azurewebsites.net`.

The production settings are specific to configuring Django to run in any production environment and aren't particular to App Service. For more information, see the [Django deployment checklist](#). Also see [Production settings for Django on Azure](#) for details on some of the changes.

Having issues? [Let us know](#).

## 3. Create Postgres database in Azure

Install the `db-up` extension for the Azure CLI:

```
az extension add --name db-up
```

If the `az` command is not recognized, be sure you have the Azure CLI installed as described in [Set up your initial environment](#).

Then create the Postgres database in Azure with the `az postgres up` command:

```
az postgres up --resource-group DjangoPostgres-tutorial-rg --location centralus --sku-name B_Gen5_1 --server-name <postgres-server-name> --database-name pollsdb --admin-user <admin-username> --admin-password <admin-password> --ssl-enforcement Enabled
```

- Replace `<postgres-server-name>` with a name that's **unique across all Azure** (the server endpoint becomes <https://<postgres-server-name>.postgres.database.azure.com>). A good pattern is to use a combination of your company name and another unique value.
- For `<admin-username>` and `<admin-password>`, specify credentials to create an administrator user for this Postgres server. The admin username can't be `azure_superuser`, `azure_pg_admin`, `admin`, `administrator`, `root`, `guest`, or `public`. It can't start with `pg_`. The password must contain **8 to 128 characters** from three of the following categories: English uppercase letters, English lowercase letters, numbers (0 through 9), and non-alphanumeric characters (for example, !, #, %). The password cannot contain username.
- Do not use the `$` character in the username or password. Later you create environment variables with these values where the `$` character has special meaning within the Linux container used to run Python apps.
- The `B_Gen5_1` (Basic, Gen5, 1 core) [pricing tier](#) used here is the least expensive. For production databases, omit the `--sku-name` argument to use the `GP_Gen5_2` (General Purpose, Gen 5, 2 cores) tier instead.

This command performs the following actions, which may take a few minutes:

- Create a [resource group](#) called `DjangoPostgres-tutorial-rg`, if it doesn't already exist.
- Create a Postgres server named by the `--server-name` argument.
- Create an administrator account using the `--admin-user` and `--admin-password` arguments. You can omit these arguments to allow the command to generate unique credentials for you.
- Create a `pollsdb` database as named by the `--database-name` argument.
- Enable access from your local IP address.
- Enable access from Azure services.
- Create a database user with access to the `pollsdb` database.

You can do all the steps separately with other `az postgres` and `psql` commands, but `az postgres up` does all the steps together.

When the command completes, it outputs a JSON object that contains different connection strings for the database along with the server URL, a generated user name (such as "joyfulKoala@msdocs-djangodb-12345"), and a GUID password. **Copy the user name and password to a temporary text file** as you need them later in this tutorial.

#### TIP

`-l <location-name>`, can be set to any one of the [Azure regions](#). You can get the regions available to your subscription with the `az account list-locations` command. For production apps, put your database and your app in the same location.

1. Enable parameters caching with the Azure CLI so you don't need to provide those parameters with every

command. (Cached values are saved in the `.azure` folder.)

```
az config param-persist on
```

2. Create a [resource group](#) (you can change the name, if desired). The resource group name is cached and automatically applied to subsequent commands.

```
az group create --name Python-Django-PGFlex-rg --location centralus
```

3. Create the database server (the process takes a few minutes):

```
az postgres flexible-server create --sku-name Standard_B1ms --public-access all
```

If the `az` command is not recognized, be sure you have the Azure CLI installed as described in [Set up your initial environment](#).

The `az postgres flexible-server create` command performs the following actions, which take a few minutes:

- Create a default resource group if there's not a cached name already.
- Create a PostgreSQL Flexible Server:
  - By default, the command uses a generated name like `server383813186`. You can specify your own name with the `--name` parameter. The name must be unique across all of Azure.
  - The command uses the lowest-cost `Standard_B1ms` pricing tier. Omit the `--sku-name` argument to use the default `Standard_D2s_V3` tier.
  - The command uses the resource group and location cached from the previous `az group create` command, which in this example is the resource group `Python-Django-PGFlex-rg` in the `centralus` region.
- Create an administrator account with a username and password. You can specify these values directly with the `--admin-user` and `--admin-password` parameters.
- Create a database named `flexibleserverdb` by default. You can specify a database name with the `--database-name` parameter.
- Enables complete public access, which you can control using the `--public-access` parameter.

4. When the command completes, [copy the command's JSON output to a file](#) as you need values from the output later in this tutorial, specifically the host, username, and password, along with the database name.

Having issues? [Let us know](#).

## 4. Deploy the code to Azure App Service

In this section, you create app host in App Service app, connect this app to the Postgres database, then deploy your code to that host.

### 4.1 Create the App Service app

In the terminal, make sure you're in the `djangoapp` repository folder that contains the app code.

Create an App Service app (the host process) with the `az webapp up` command:

```
az webapp up --resource-group DjangoPostgres-tutorial-rg --location centralus --plan DjangoPostgres-tutorial-plan --sku B1 --name <app-name>
```

- For the `--location` argument, use the same location as you did for the database in the previous section.
- Replace `<app-name>` with a unique name across all Azure (the server endpoint is `https://<app-name>.azurewebsites.net`). Allowed characters for `<app-name>` are A - Z, 0 - 9, and -. A good pattern is to use a combination of your company name and an app identifier.

This command performs the following actions, which may take a few minutes:

- Create the [resource group](#) if it doesn't already exist. (In this command you use the same resource group in which you created the database earlier.)
- Create the [App Service plan](#) `DjangoPostgres-tutorial-plan` in the Basic pricing tier (B1), if it doesn't exist. `--plan` and `--sku` are optional.
- Create the App Service app if it doesn't exist.
- Enable default logging for the app, if not already enabled.
- Upload the repository using ZIP deployment with build automation enabled.
- Cache common parameters, such as the name of the resource group and App Service plan, into the file `.azure/config`. As a result, you don't need to specify all the same parameter with later commands. For example, to redeploy the app after making changes, you can just run `az webapp up` again without any parameters. Commands that come from CLI extensions, such as `az postgres up`, however, do not at present use the cache, which is why you needed to specify the resource group and location here with the initial use of `az webapp up`.

1. In the terminal, make sure you're in the `djangapp` repository folder that contains the app code.
2. Switch to the sample app's `flexible-server` branch. This branch contains specific configuration needed for PostgreSQL Flexible Server:

```
git checkout flexible-server
```

3. Run the following `az webapp up` command to create the App Service host for the app:

```
az webapp up --name <app-name> --sku B1
```

This command performs the following actions, which may take a few minutes, using resource group and location cached from the previous `az group create` command (the group `Python-Django-PGFlex-rg` in the `centralus` region in this example).

- Create an [App Service plan](#) in the Basic pricing tier (B1). You can omit `--sku` to use default values.
- Create the App Service app.
- Enable default logging for the app.
- Upload the repository using ZIP deployment with build automation enabled.

Upon successful deployment, the command generates JSON output like the following example:

```
{
 "URL": "http://msdocs-djangoapp-12345.azurewebsites.net",
 "appserviceplan": "DjangoPostgres-tutorial-plan",
 "location": "westus",
 "name": "msdocs-djangoapp-12345",
 "os": "Linux",
 "resourcegroup": "DjangoPostgres-tutorial-rg",
 "runtime_version": "python|3.7",
 "runtime_version_detected": "-",
 "sku": "BASIC",
 "src_path": "//var//lib//postgresql//djangoapp"
}
```

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

## 4.2 Configure environment variables to connect the database

With the code now deployed to App Service, the next step is to connect the app to the Postgres database in Azure.

The app code expects to find database information in four environment variables named `DBHOST`, `DBNAME`, `DBUSER`, and `DBPASS`.

To set environment variables in App Service, create "app settings" with the following [az webapp config appsettings set](#) command.

```
az webapp config appsettings set --settings DBHOST=<postgres-server-name> DBUSER=<username> DBPASS=<password> DBNAME="pollsdb"
```

- Replace `<postgres-server-name>` with the name you used earlier with the `az postgres up` command. The code in `azuresite/production.py` automatically appends `.postgres.database.azure.com` to create the full Postgres server URL.
- Replace `<username>` and `<password>` with the administrator credentials that you used with the earlier `az postgres up` command, or those that `az postgres up` generated for you. The code in `azuresite/production.py` automatically constructs the full Postgres username from `DBUSER` and `DBHOST`, so don't include the `@server` portion. (Also, as noted earlier, you should not use the `$` character in either value as it has a special meaning for Linux environment variables.)
- The resource group and app names are drawn from the cached values in the `.azure/config` file.

```
az webapp config appsettings set --settings DBHOST=<host> DBUSER=<username> DBPASS=<password> DBNAME="flexibleserverdb"
```

Replace the host, username, and password values with those from the output of the `az postgres flexible-server create` command used earlier. The host should be a URL like `server383813186.postgres.database.azure.com`.

Also replace `flexibleserverdb` with the database name if you changed it with the `az postgres flexible-server create` command.

In your Python code, you access these settings as environment variables with statements like `os.environ.get('DBHOST')`. For more information, see [Access environment variables](#).

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

## 4.3 Run Django database migrations

Django database migrations ensure that the schema in the PostgreSQL on Azure database match those described in your code.

1. Run `az webapp ssh` to open an SSH session for the web app in the browser:

```
az webapp ssh
```

If you cannot connect to the SSH session, then the app itself has failed to start. [Check the diagnostic logs](#) for details. For example, if you haven't created the necessary app settings in the previous section, the logs will indicate `KeyError: 'DBNAME'`.

2. In the SSH session, run the following commands (you can paste commands using **Ctrl+Shift+V**):

```
Run database migrations
python manage.py migrate

Create the super user (follow prompts)
python manage.py createsuperuser
```

If you encounter any errors related to connecting to the database, check the values of the application settings created in the previous section.

3. The `createsuperuser` command prompts you for superuser credentials. For the purposes of this tutorial, use the default username `root`, press **Enter** for the email address to leave it blank, and enter `Pollsdb1` for the password.
4. If you see an error that the database is locked, make sure that you ran the `az webapp settings` command in the previous section. Without those settings, the migrate command cannot communicate with the database, resulting in the error.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

#### 4.4 Create a poll question in the app

1. Open the app website. The app should display the message "Polls app" and "No polls are available" because there are no specific polls yet in the database.

```
az webapp browse
```

If you see "Application Error", then it's likely that you either didn't create the required settings in the previous step, [Configure environment variables to connect the database](#), or that those value contain errors. Run the command `az webapp config appsettings list` to check the settings. You can also [check the diagnostic logs](#) to see specific errors during app startup. For example, if you didn't create the settings, the logs will show the error, `KeyError: 'DBNAME'`.

After updating the settings to correct any errors, give the app a minute to restart, then refresh the browser.

2. Browse to the web app's admin page by appending `/admin` to the URL, for example, `http://<app-name>.azurewebsites.net/admin`. Sign in using Django superuser credentials from the previous section (`root` and `Pollsdb1`). Under **Polls**, select **Add** next to **Questions** and create a poll question with some choices.
3. Return to the main the website (`http://<app-name>.azurewebsites.net`) to confirm that the questions are now presented to the user. Answer questions however you like to generate some data in the database.

**Congratulations!** You're running a Python Django web app in Azure App Service for Linux, with an active Postgres database.

**NOTE**

App Service detects a Django project by looking for a `wsgi.py` file in each subfolder, which `manage.py startproject` creates by default. When App Service finds that file, it loads the Django web app. For more information, see [Configure built-in Python image](#).

## 5. Make code changes and redeploy

In this section, you make local changes to the app and redeploy the code to App Service. In the process, you set up a Python virtual environment that supports ongoing work.

### 5.1 Run the app locally

In a terminal window, run the following commands. Be sure to follow the prompts when creating the superuser:

- [bash](#)
- [PowerShell](#)
- [CMD](#)

```
Configure the Python virtual environment
python3 -m venv venv
source venv/bin/activate

Install dependencies
pip install -r requirements.txt
Run Django migrations
python manage.py migrate
Create Django superuser (follow prompts)
python manage.py createsuperuser
Run the dev server
python manage.py runserver
```

Once the web app is fully loaded, the Django development server provides the local app URL in the message, "Starting development server at <http://127.0.0.1:8000/>. Quit the server with CTRL-BREAK".

```
Performing system checks...

System check identified no issues (0 silenced).
June 24, 2020 - 10:27:14
Django version 2.2.13, using settings 'azuresite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Test the app locally with the following steps:

1. Go to `http://localhost:8000` in a browser, which should display the message "No polls are available".
2. Go to `http://localhost:8000/admin` and sign in using the admin user you created previously. Under **Polls**, again select **Add** next to **Questions** and create a poll question with some choices.
3. Go to `http://localhost:8000` again and answer the question to test the app.
4. Stop the Django server by pressing **Ctrl+C**.

When running locally, the app is using a local Sqlite3 database and doesn't interfere with your production database. You can also use a local PostgreSQL database, if desired, to better simulate your production environment.

Having issues? [Let us know](#).

## 5.2 Update the app

In `polls/models.py`, locate the line that begins with `choice_text` and change the `max_length` parameter to 100:

```
Find this line of code and set max_length to 100 instead of 200
choice_text = models.CharField(max_length=100)
```

Because you changed the data model, create a new Django migration and migrate the database:

```
python manage.py makemigrations
python manage.py migrate
```

Run the development server again with `python manage.py runserver` and test the app at to `http://localhost:8000/admin`:

Stop the Django web server again with `Ctrl+C`.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

## 5.3 Redeploy the code to Azure

Run the following command in the repository root:

```
az webapp up
```

This command uses the parameters cached in the `.azure/config` file. Because App Service detects that the app already exists, it just redeploys the code.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

## 5.4 Rerun migrations in Azure

Because you made changes to the data model, you need to rerun database migrations in App Service.

Open an SSH session again in the browser by navigating to

`https://<app-name>.scm.azurewebsites.net/webssh/host`. Then run the following command:

```
python manage.py migrate
```

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

## 5.5 Review app in production

Browse to the app again (using `az webapp browse` or navigating to `http://<app-name>.azurewebsites.net`) and test the app again in production. (Because you changed only the length of a database field, the change is only noticeable if you try to enter a longer response when creating a question.)

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

# 6. Stream diagnostic logs

You can access the console logs generated from inside the container that hosts the app on Azure.

Run the following Azure CLI command to see the log stream. This command uses parameters cached in the `.azure/config` file.

```
az webapp log tail
```

If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type **Ctrl +C**.

Having issues? [Let us know](#).

#### NOTE

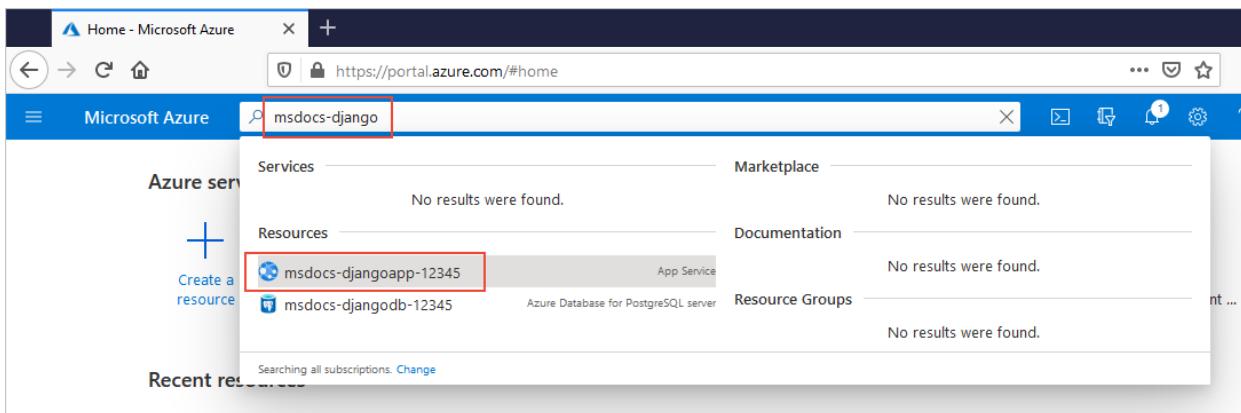
You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

`az webapp up` turns on the default logging for you. For performance reasons, this logging turns itself off after some time, but turns back on each time you run `az webapp up` again. To turn it on manually, run the following command:

```
az webapp log config --docker-container-logging filesystem
```

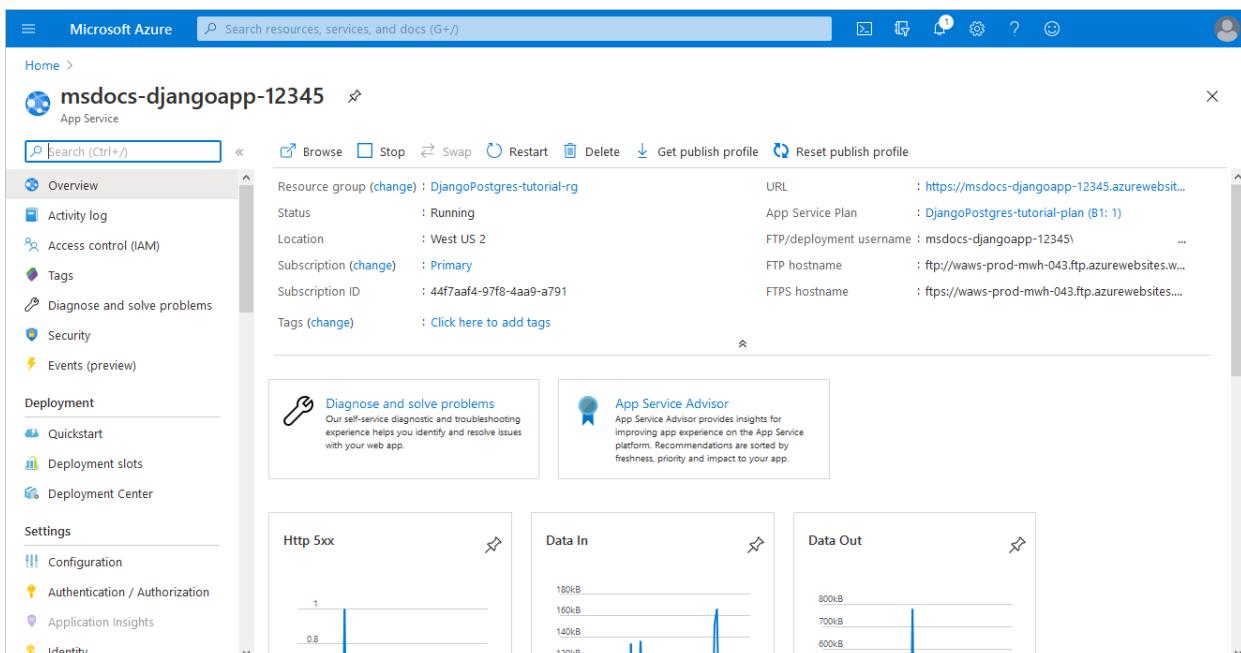
## 7. Manage your app in the Azure portal

In the [Azure portal](#), search for the app name and select the app in the results.



The screenshot shows the Microsoft Azure portal's search interface. The search bar at the top contains the text "msdocs-django". Below the search bar, the results are displayed under the "Resources" category. A single result, "msdocs-djangoapp-12345", is shown with its icon and name. This result is highlighted with a red box. Other resources listed include "msdocs-djangodb-12345" and "Azure Database for PostgreSQL server". The "Services" and "Marketplace" sections show no results.

By default, the portal shows your app's **Overview** page, which provides a general performance view. Here, you can also perform basic management tasks like browse, stop, restart, and delete. The tabs on the left side of the page show the different configuration pages you can open.



The screenshot shows the Microsoft Azure portal's Overview page for the app "msdocs-djangoapp-12345". The top navigation bar includes the Microsoft Azure logo, a search bar, and various icons. The main content area has a header with the app name and a "Search (Ctrl+/" input field. Below the header, there are several tabs: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Events (preview), Deployment, Quickstart, Deployment slots, Deployment Center, Settings, Configuration, Authentication / Authorization, Application Insights, and Identity. The "Overview" tab is selected. Key details shown include the Resource group ("DjangoPostgres-tutorial-rg"), Status ("Running"), Location ("West US 2"), Subscription ("Primary"), Subscription ID ("44f7aaaf4-97f8-4aa9-a791"), and Tags ("Click here to add tags"). On the right, there are sections for "Diagnose and solve problems" (with a link to "Self-service diagnostic and troubleshooting"), "App Service Advisor" (with a link to "Provides insights for improving app experience on the App Service platform"), and performance metrics: "Http 5xx" (1), "Data In" (180kB, 160kB, 140kB, 120kB), and "Data Out" (800kB, 700kB, 600kB). The overall layout is clean and modern, typical of the Azure portal's design.

Having issues? Refer first to the [Troubleshooting guide](#), otherwise, [let us know](#).

## 8. Clean up resources

If you'd like to keep the app or continue to the additional tutorials, skip ahead to [Next steps](#). Otherwise, to avoid incurring ongoing charges you can delete the resource group created for this tutorial:

```
az group delete --name Python-Django-PGFlex-rg --no-wait
```

By deleting the resource group, you also deallocate and delete all the resources contained within it. Be sure you no longer need the resources in the group before using the command.

Deleting all the resources can take some time. The `--no-wait` argument allows the command to return immediately.

Having issues? [Let us know](#).

## Next steps

Learn how to map a custom DNS name to your app:

[Tutorial: Map custom DNS name to your app](#)

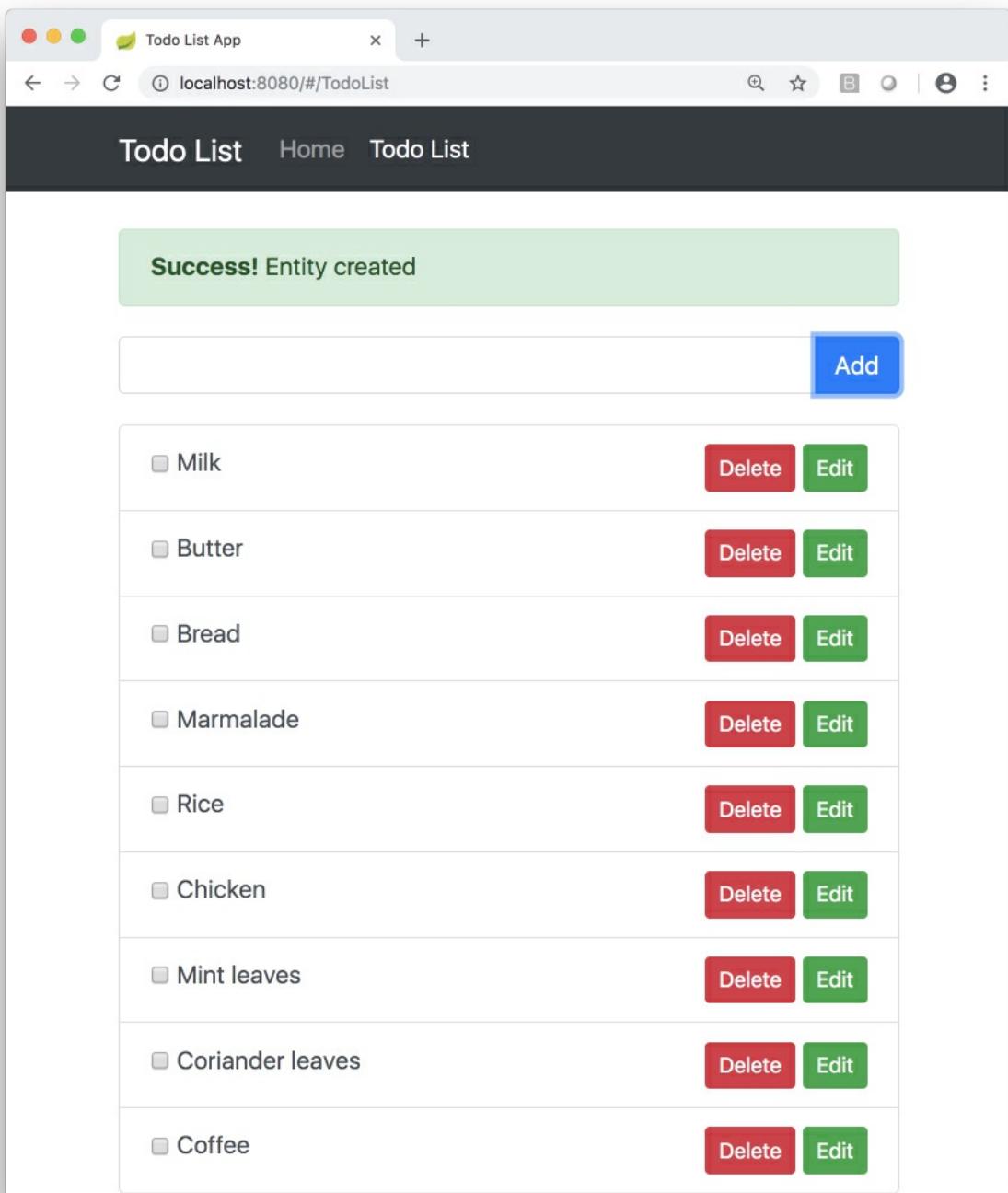
Learn how App Service runs a Python app:

[Configure Python app](#)

# Tutorial: Build a Java Spring Boot web app with Azure App Service on Linux and Azure Cosmos DB

11/2/2021 • 6 minutes to read • [Edit Online](#)

This tutorial walks you through the process of building, configuring, deploying, and scaling Java web apps on Azure. When you are finished, you will have a [Spring Boot](#) application storing data in [Azure Cosmos DB](#) running on [Azure App Service on Linux](#).



In this tutorial, you learn how to:

- Create a Cosmos DB database.
- Connect a sample app to the database and test it locally

- Deploy the sample app to Azure
- Stream diagnostic logs from App Service
- Add additional instances to scale out the sample app

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- [Azure CLI](#), installed on your own computer.
- [Git](#)
- [Java JDK](#)
- [Maven](#)

## Clone the sample TODO app and prepare the repo

This tutorial uses a sample TODO list app with a web UI that calls a Spring REST API backed by [Spring Data Azure Cosmos DB](#). The code for the app is available [on GitHub](#). To learn more about writing Java apps using Spring and Cosmos DB, see the [Spring Boot Starter with the Azure Cosmos DB SQL API tutorial](#) and the [Spring Data Azure Cosmos DB quick start](#).

Run the following commands in your terminal to clone the sample repo and set up the sample app environment.

```
git clone --recurse-submodules https://github.com/Azure-Samples/e2e-java-experience-in-app-service-linux-part-2.git
cd e2e-java-experience-in-app-service-linux-part-2
yes | cp -rf .prep/* .
```

## Create an Azure Cosmos DB

Follow these steps to create an Azure Cosmos DB database in your subscription. The TODO list app will connect to this database and store its data when running, persisting the application state no matter where you run the application.

1. Login to your Azure CLI, and optionally set your subscription if you have more than one connected to your login credentials.

```
az login
az account set -s <your-subscription-id>
```

2. Create an Azure Resource Group, noting the resource group name.

```
az group create -n <your-azure-group-name> \
-l <your-resource-group-region>
```

3. Create Azure Cosmos DB with the `GlobalDocumentDB` kind. The name of Cosmos DB must use only lower case letters. Note down the `documentEndpoint` field in the response from the command.

```
az cosmosdb create --kind GlobalDocumentDB \
-g <your-azure-group-name> \
-n <your-azure-COSMOS-DB-name-in-lower-case-letters>
```

4. Get your Azure Cosmos DB key to connect to the app. Keep the `primaryMasterKey`, `documentEndpoint` nearby as you'll need them in the next step.

```
az cosmosdb keys list -g <your-azure-group-name> -n <your-azure-COSMOSDB-name>
```

## Configure the TODO app properties

Open a terminal on your computer. Copy the sample script file in the cloned repo so you can customize it for your Cosmos DB database you just created.

```
cd initial/spring-todo-app
cp set-env-variables-template.sh .scripts/set-env-variables.sh
```

Edit `.scripts/set-env-variables.sh` in your favorite editor and supply Azure Cosmos DB connection info. For the App Service Linux configuration, use the same region as before (`your-resource-group-region`) and resource group (`your-azure-group-name`) used when creating the Cosmos DB database. Choose a `WEBAPP_NAME` that is unique since it cannot duplicate any web app name in any Azure deployment.

```
export COSMOSDB_URI=<put-your-COSMOS-DB-documentEndpoint-URI-here>
export COSMOSDB_KEY=<put-your-COSMOS-DB-primaryMasterKey-here>
export COSMOSDB_DBNAME=<put-your-COSMOS-DB-name-here>

App Service Linux Configuration
export RESOURCEGROUP_NAME=<put-your-resource-group-name-here>
export WEBAPP_NAME=<put-your-Webapp-name-here>
export REGION=<put-your-REGION-here>
```

Then run the script:

```
source .scripts/set-env-variables.sh
```

These environment variables are used in `application.properties` in the TODO list app. The fields in the properties file set up a default repository configuration for Spring Data:

```
azure.cosmosdb.uri=${COSMOSDB_URI}
azure.cosmosdb.key=${COSMOSDB_KEY}
azure.cosmosdb.database=${COSMOSDB_DBNAME}
```

```
@Repository
public interface TodoItemRepository extends DocumentDbRepository<TodoItem, String> {
}
```

Then the sample app uses the `@Document` annotation imported from `com.microsoft.azure.spring.data.cosmosdb.core.mapping.Document` to set up an entity type to be stored and managed by Cosmos DB:

```
@Document
public class TodoItem {
 private String id;
 private String description;
 private String owner;
 private boolean finished;
```

## Run the sample app

Use Maven to run the sample.

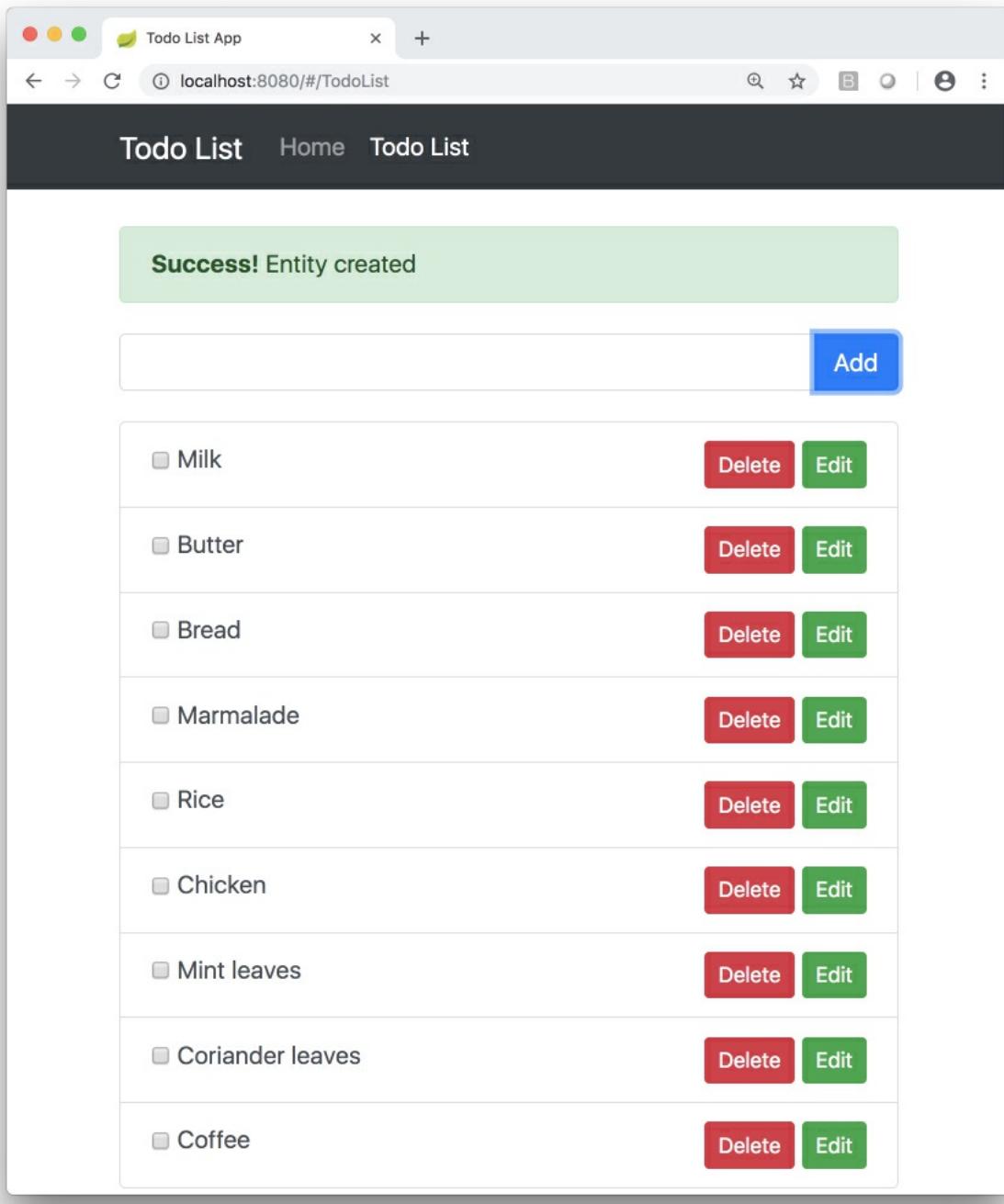
```
mvn package spring-boot:run
```

The output should look like the following.

```
bash-3.2$ mvn package spring-boot:run
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building spring-todo-app 2.0-SNAPSHOT
[INFO] -----
[INFO]

[INFO] SimpleUrlHandlerMapping - Mapped URL path [/webjars/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[INFO] SimpleUrlHandlerMapping - Mapped URL path [/**] onto handler of type [class
org.springframework.web.servlet.resource.ResourceHttpRequestHandler]
[INFO] WelcomePageHandlerMapping - Adding welcome page: class path resource [static/index.html]
2018-10-28 15:04:32.101 INFO 7673 --- [main] c.m.azure.documentdb.DocumentClient :
Initializing DocumentClient with serviceEndpoint [https://sample-cosmos-db-westus.documents.azure.com:443/],
ConnectionPolicy [ConnectionPolicy [requestTimeout=60, mediaRequestTimeout=300, connectionMode=Gateway,
mediaReadMode=Buffered, maxPoolSize=800, idleConnectionTimeout=60, userAgentSuffix=spring-
data/2.0.6;098063be661ab767976bd5a2ec350e978faba99348207e8627375e8033277cb2,
retryOptions=com.microsoft.azure.documentdb.RetryOptions@6b9fb84d, enableEndpointDiscovery=true,
preferredLocations=null]], ConsistencyLevel [null]
[INFO] AnnotationMBeanExporter - Registering beans for JMX exposure on startup
[INFO] TomcatWebServer - Tomcat started on port(s): 8080 (http) with context path ''
[INFO] TodoApplication - Started TodoApplication in 45.573 seconds (JVM running for 76.534)
```

You can access Spring TODO App locally using this link once the app is started: <http://localhost:8080/>.



If you see exceptions instead of the "Started TodoApplication" message, check that the `bash` script in the previous step exported the environment variables properly and that the values are correct for the Azure Cosmos DB database you created.

## Configure Azure deployment

Open the `pom.xml` file in the `initial/spring-boot-todo` directory and add the following [Azure Web App Plugin for Maven](#) configuration.

```

<plugins>

 <!--*****-->
 <!-- Deploy to Java SE in App Service Linux -->
 <!--*****-->

 <plugin>
 <groupId>com.microsoft.azure</groupId>
 <artifactId>azure-webapp-maven-plugin</artifactId>
 <version>2.0.0</version>
 <configuration>
 <schemaVersion>v2</schemaVersion>

 <!-- Web App information -->
 <resourceGroup>${RESOURCEGROUP_NAME}</resourceGroup>
 <appName>${WEBAPP_NAME}</appName>
 <region>${REGION}</region>
 <pricingTier>P1v2</pricingTier>
 <!-- Java Runtime Stack for Web App on Linux-->
 <runtime>
 <os>linux</os>
 <javaVersion>Java 8</javaVersion>
 <webContainer>Java SE</webContainer>
 </runtime>
 <deployment>
 <resources>
 <resource>
 <directory>${project.basedir}/target</directory>
 <includes>
 <include>*.jar</include>
 </includes>
 </resource>
 </resources>
 </deployment>

 <appSettings>
 <property>
 <name>COSMOSDB_URI</name>
 <value>${COSMOSDB_URI}</value>
 </property>
 <property>
 <name>COSMOSDB_KEY</name>
 <value>${COSMOSDB_KEY}</value>
 </property>
 <property>
 <name>COSMOSDB_DBNAME</name>
 <value>${COSMOSDB_DBNAME}</value>
 </property>
 <property>
 <name>JAVA_OPTS</name>
 <value>-Dserver.port=80</value>
 </property>
 </appSettings>

 </configuration>
 </plugin>
 ...
 </plugins>

```

## Deploy to App Service on Linux

Use the `mvn azure-webapp:deploy` Maven goal to deploy the TODO app to Azure App Service on Linux.

```
Deploy
bash-3.2$ mvn azure-webapp:deploy
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building spring-todo-app 2.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- azure-webapp-maven-plugin:2.0.0:deploy (default-cli) @ spring-todo-app ---
Auth Type: AZURE_CLI
Default subscription: xxxxxxxxx
Username: xxxxxxxxx
[INFO] Subscription: xxxxxxxxx
[INFO] Creating App Service Plan 'ServicePlanb6ba8178-5bbb-49e7'...
[INFO] Successfully created App Service Plan.
[INFO] Creating web App spring-todo-app...
[INFO] Successfully created Web App spring-todo-app.
[INFO] Trying to deploy artifact to spring-todo-app...
[INFO] Successfully deployed the artifact to https://spring-todo-app.azurewebsites.net
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 02:19 min
[INFO] Finished at: 2019-11-06T15:32:03-07:00
[INFO] Final Memory: 50M/574M
[INFO] -----
```

The output contains the URL to your deployed application (in this example,

<https://spring-todo-app.azurewebsites.net>). You can copy this URL into your web browser or run the following command in your Terminal window to load your app.

```
explorer https://spring-todo-app.azurewebsites.net
```

You should see the app running with the remote URL in the address bar:

The screenshot shows a web browser window titled "Todo List App" with the URL "https://spring-todo-app.azurewebsites.net/#/TodoList". The page has a header with "Todo List" and "Home" buttons. Below the header is a search bar and a "Add" button. The main content area displays a list of items:

Item	Action	Action
Milk	Delete	Edit
Butter	Delete	Edit
Bread	Delete	Edit
Marmalade	Delete	Edit
Rice	Delete	Edit
Chicken	Delete	Edit
Mint leaves	Delete	Edit
Coriander leaves	Delete	Edit
Coffee	Delete	Edit

## Stream diagnostic logs

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

**NOTE**

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

## Scale out the TODO App

Scale out the application by adding another worker:

```
az appservice plan update --number-of-workers 2 \
--name ${WEBAPP_PLAN_NAME} \
--resource-group <your-azure-group-name>
```

## Clean up resources

If you don't need these resources for another tutorial (see [Next steps](#)), you can delete them by running the following command in the Cloud Shell:

```
az group delete --name <your-azure-group-name> --yes
```

## Next steps

[Azure for Java Developers Spring Boot](#), [Spring Data for Cosmos DB](#), [Azure Cosmos DB](#) and [App Service Linux](#).

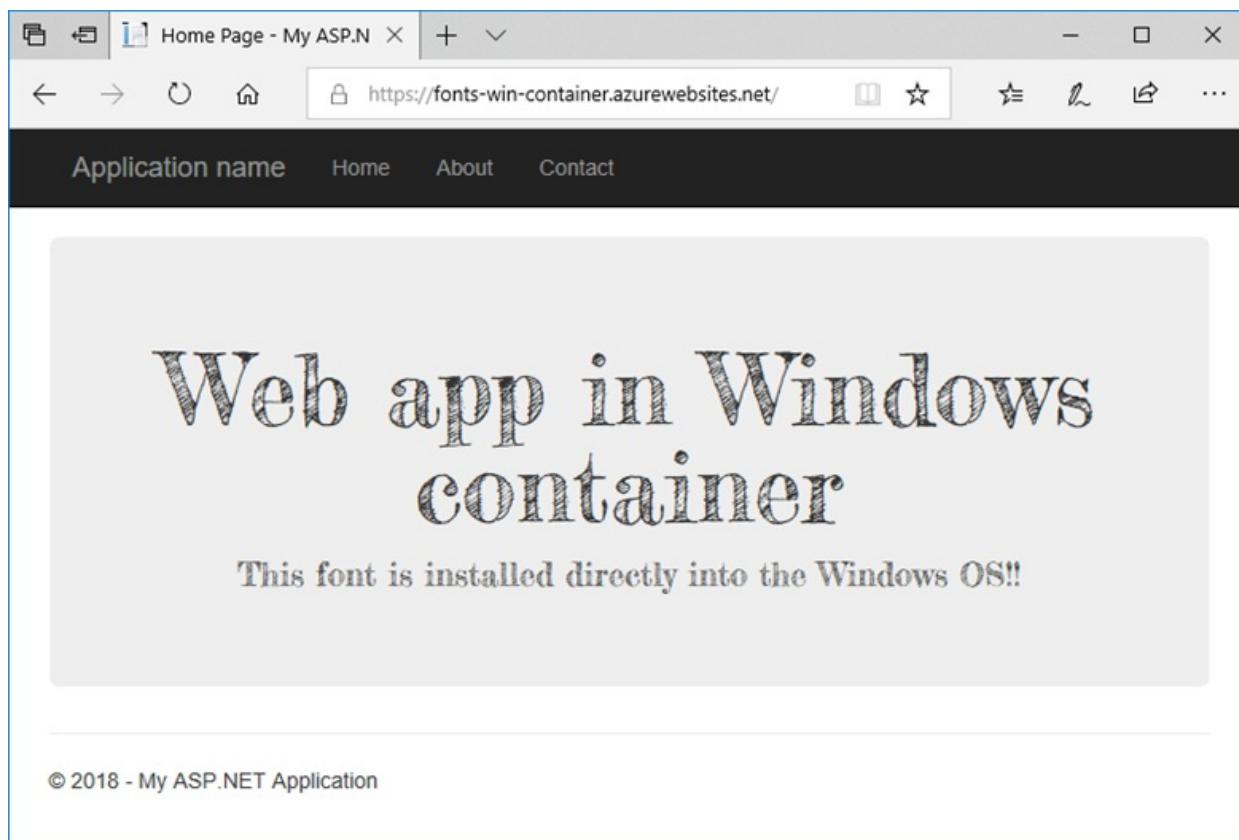
Learn more about running Java apps on App Service on Linux in the developer guide.

[Java in App Service Linux dev guide](#)

# Migrate custom software to Azure App Service using a custom container

11/2/2021 • 19 minutes to read • [Edit Online](#)

Azure App Service provides pre-defined application stacks on Windows like ASP.NET or Node.js, running on IIS. The preconfigured Windows environment locks down the operating system from administrative access, software installations, changes to the global assembly cache, and so on (see [Operating system functionality on Azure App Service](#)). However, using a custom Windows container in App Service lets you make OS changes that your app needs, so it's easy to migrate on-premises app that requires custom OS and software configuration. This tutorial demonstrates how to migrate to App Service an ASP.NET app that uses custom fonts installed in the Windows font library. You deploy a custom-configured Windows image from Visual Studio to [Azure Container Registry](#), and then run it in App Service.



## Prerequisites

To complete this tutorial:

- [Sign up for a Docker Hub account](#)
- [Install Docker for Windows](#).
- [Switch Docker to run Windows containers](#).
- [Install Visual Studio 2019](#) with the **ASP.NET and web development** and **Azure development** workloads.  
If you've installed Visual Studio 2019 already:
  - Install the latest updates in Visual Studio by clicking **Help > Check for Updates**.
  - Add the workloads in Visual Studio by clicking **Tools > Get Tools and Features**.

# Set up the app locally

## Download the sample

In this step, you set up the local .NET project.

- [Download the sample project.](#)
- Extract (unzip) the *custom-font-win-container.zip* file.

The sample project contains a simple ASP.NET application that uses a custom font that is installed into the Windows font library. It's not necessary to install fonts, but it's an example of an app that is integrated with the underlying OS. To migrate such an app to App Service, you either rearchitect your code to remove the integration, or migrate it as-is in a custom Windows container.

## Install the font

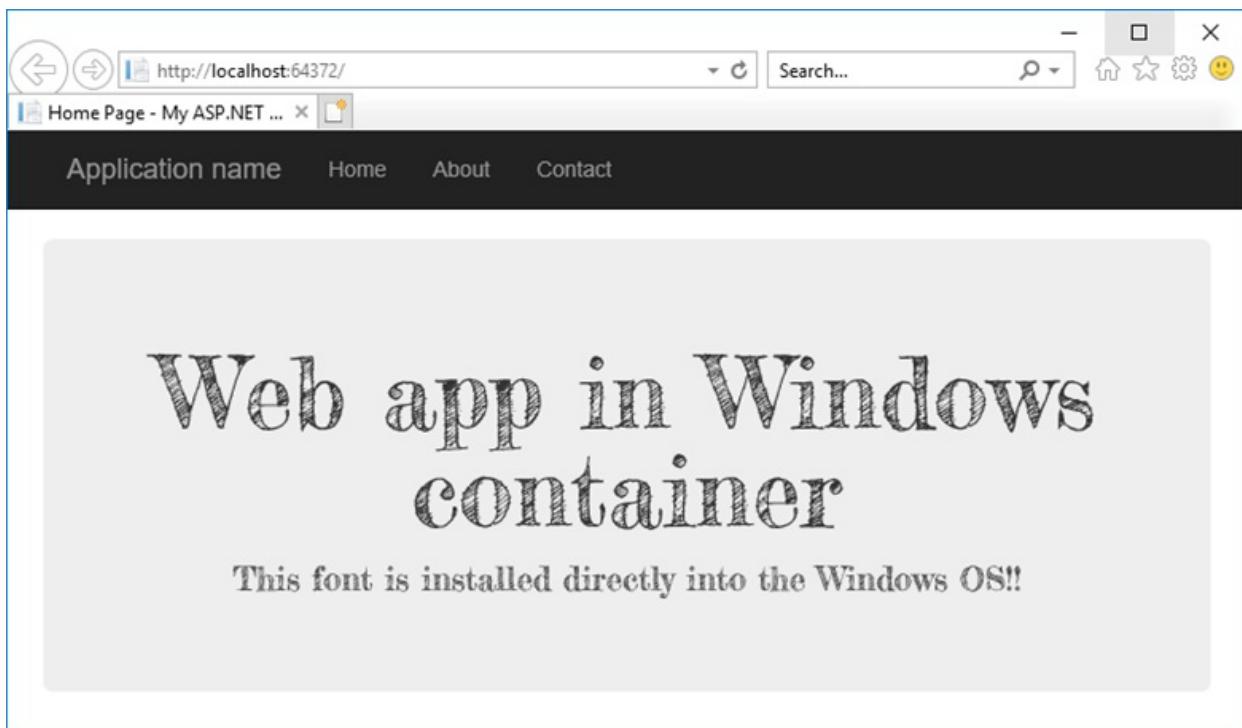
In Windows Explorer, navigate to *custom-font-win-container-master/CustomFontSample*, right-click *FredericktheGreat-Regular.ttf*, and select **Install**.

This font is publicly available from [Google Fonts](#).

## Run the app

Open the *custom-font-win-container/CustomFontSample.sln* file in Visual Studio.

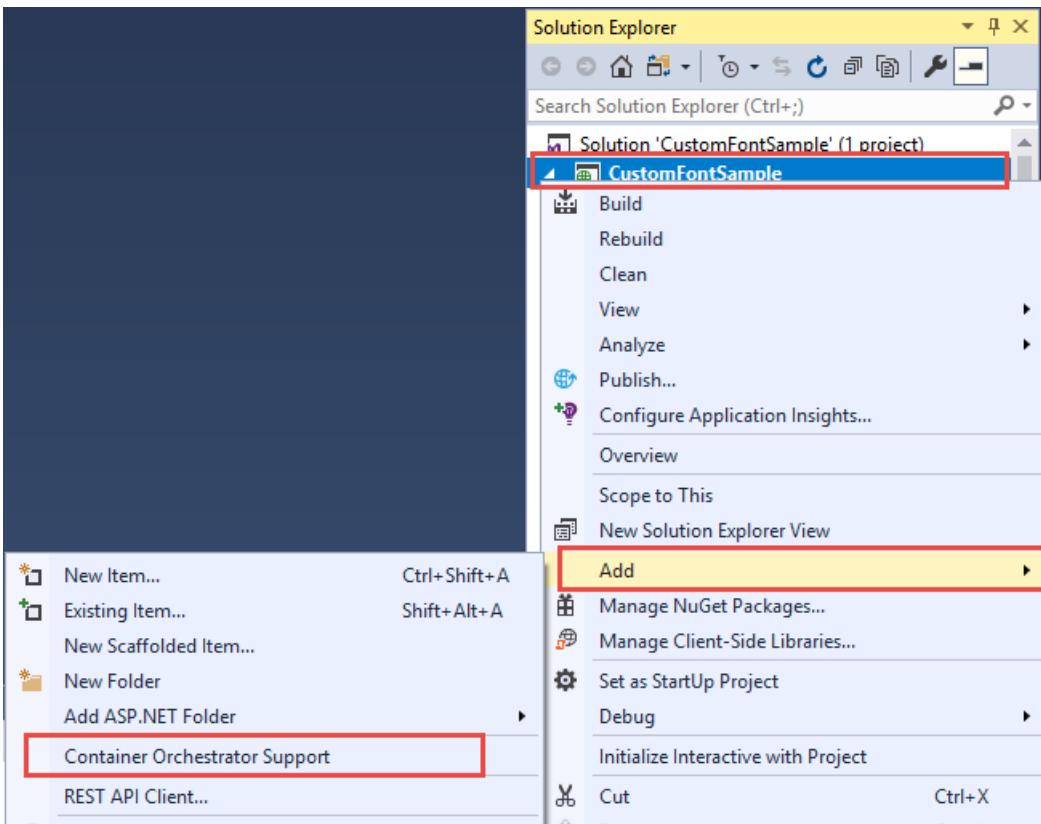
Type `Ctrl+F5` to run the app without debugging. The app is displayed in your default browser.



Because it uses an installed font, the app can't run in the App Service sandbox. However, you can deploy it using a Windows container instead, because you can install the font in the Windows container.

## Configure Windows container

In Solution Explorer, right-click the **CustomFontSample** project and select **Add > Container Orchestration Support**.



Select Docker Compose > OK.

Your project is now set up to run in a Windows container. A *Dockerfile* is added to the **CustomFontSample** project, and a **docker-compose** project is added to the solution.

From the Solution Explorer, open **Dockerfile**.

You need to use a [supported parent image](#). Change the parent image by replacing the `FROM` line with the following code:

```
FROM mcr.microsoft.com/dotnet/framework/aspnet:4.7.2-windowsservercore-1tsc2019
```

At the end of the file, add the following line and save the file:

```
RUN ${source:-obj/Docker/publish/InstallFont.ps1}
```

You can find *InstallFont.ps1* in the **CustomFontSample** project. It's a simple script that installs the font. You can find a more complex version of the script in the [Script Center](#).

#### NOTE

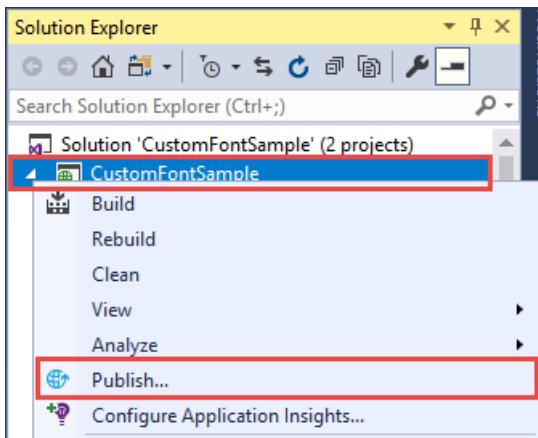
To test the Windows container locally, make sure that Docker is started on your local machine.

## Publish to Azure Container Registry

[Azure Container Registry](#) can store your images for container deployments. You can configure App Service to use images hosted in Azure Container Registry.

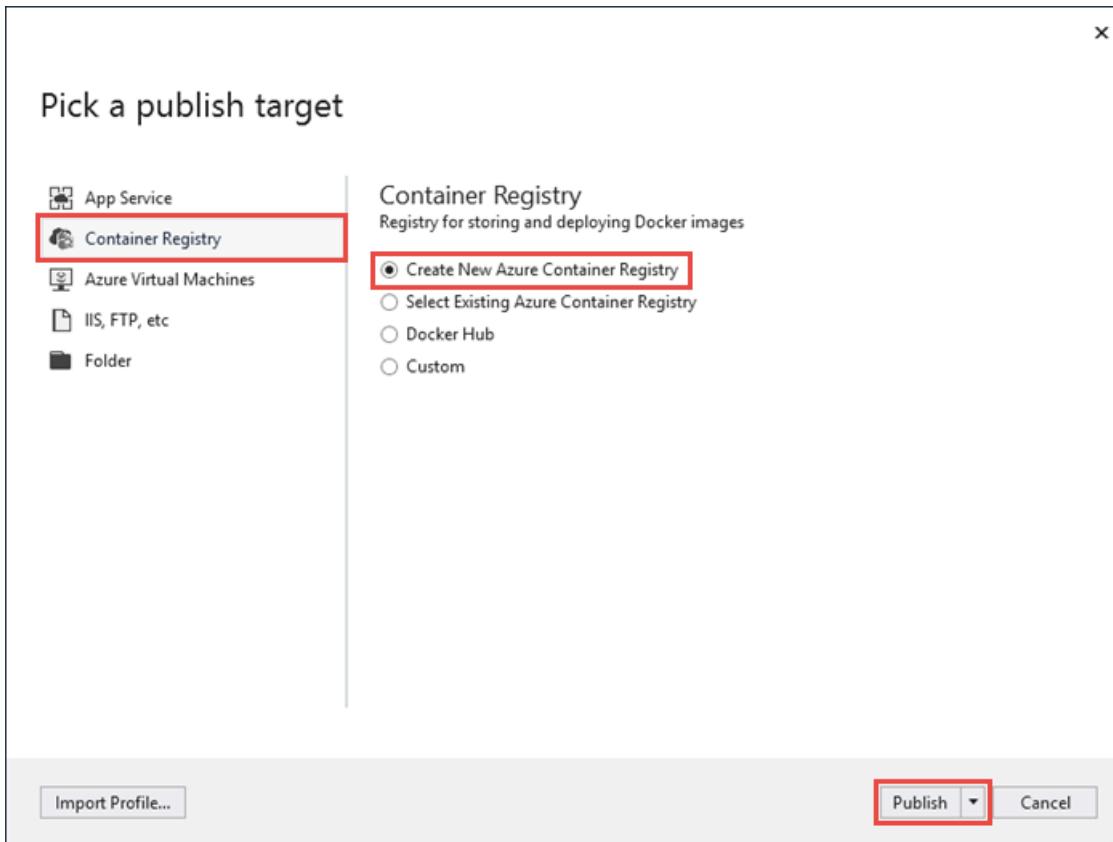
### Open publish wizard

In the Solution Explorer, right-click the **CustomFontSample** project and select **Publish**.



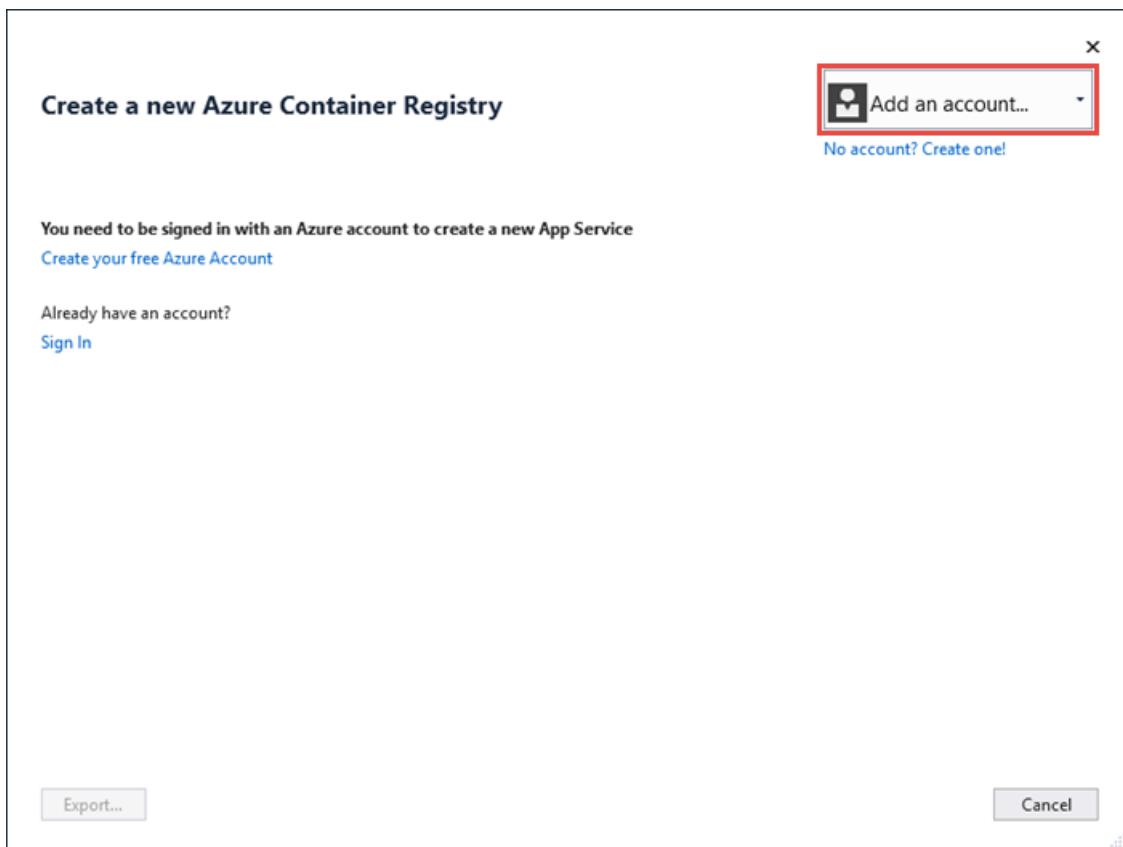
## Create registry and publish

In the publish wizard, select Container Registry > Create New Azure Container Registry > Publish.



## Sign in with Azure account

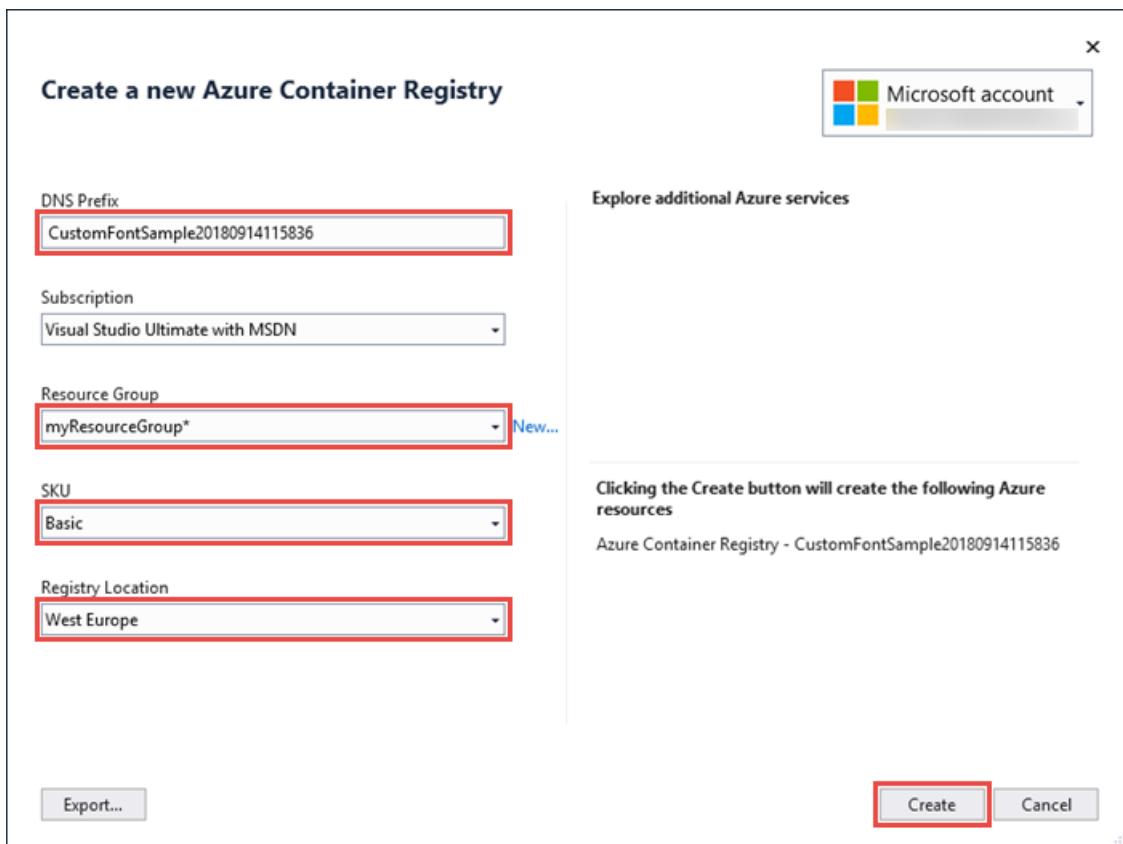
In the **Create a new Azure Container Registry** dialog, select **Add an account**, and sign in to your Azure subscription. If you're already signed in, select the account containing the desired subscription from the dropdown.



## Configure the registry

Configure the new container registry based on the suggested values in the following table. When finished, click **Create**.

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
DNS Prefix	Keep the generated registry name, or change it to another unique name.	
Resource Group	Click <b>New</b> , type <b>myResourceGroup</b> , and click <b>OK</b> .	
SKU	Basic	<a href="#">Pricing tiers</a>
Registry Location	West Europe	



A terminal window is opened and displays the image deployment progress. Wait for the deployment to complete.

## Sign in to Azure

Sign in to the Azure portal at <https://portal.azure.com>.

## Create a web app

From the left menu, select **Create a resource > Web > Web App for Containers**.

### Configure app basics

In the **Basics** tab, configure the settings according to the following table, then click **Next: Docker**.

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
Subscription	Make sure the correct subscription is selected.	
Resource Group	Select <b>Create new</b> , type <b>myResourceGroup</b> , and click <b>OK</b> .	
Name	Type a unique name.	The URL of the web app is <code>https://&lt;app-name&gt;.azurewebsites.net</code> , where <code>&lt;app-name&gt;</code> is your app name.
Publish	Docker container	
Operating System	Windows	

SETTING	SUGGESTED VALUE	FOR MORE INFORMATION
Region	West Europe	
Windows Plan	Select Create new, type myAppServicePlan, and click OK.	

Your **Basics** tab should look like this:

Basics Docker Monitoring Tags Review + create

App Service Web Apps lets you quickly build, deploy, and scale enterprise-grade web, mobile, and API apps running on any platform. Meet rigorous performance, scalability, security and compliance requirements while using a fully managed platform to perform infrastructure maintenance. [Learn more](#)

**Project Details**

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ Visual Studio Ultimate with MSDN

Resource Group \* ⓘ (New) myResourceGroup [Create new](#)

**Instance Details**

Name \* custom-font-app .azurewebsites.net

Publish \* [Code](#) [Docker Container](#)

Operating System \* [Linux](#) [Windows](#)

Region \* West Europe  ⓘ Can't find your App Service Plan, try a different region.

**App Service Plan**

App Service plan pricing tier determines the location, features, cost and compute resources associated with your app. [Learn more](#)

Windows Plan (West Europe) \* ⓘ (New) myAppServicePlan [Create new](#)

Sku and size \* Premium Container PC2  
320 total ACU, 8 GB memory [Change size](#)

[Review + create](#) < Previous Next : Docker >

## Configure Windows container

In the **Docker** tab, configure your custom Windows container as shown in the following table, and select **Review + create**.

SETTING	SUGGESTED VALUE
Image Source	Azure Container Register

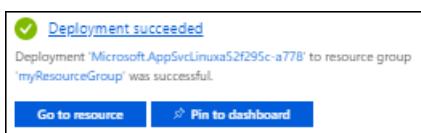
SETTING	SUGGESTED VALUE
Registry	Select the registry you created earlier.
Image	customfontsample
Tag	latest

## Complete app creation

Click **Create** and wait for Azure to create the required resources.

## Browse to the web app

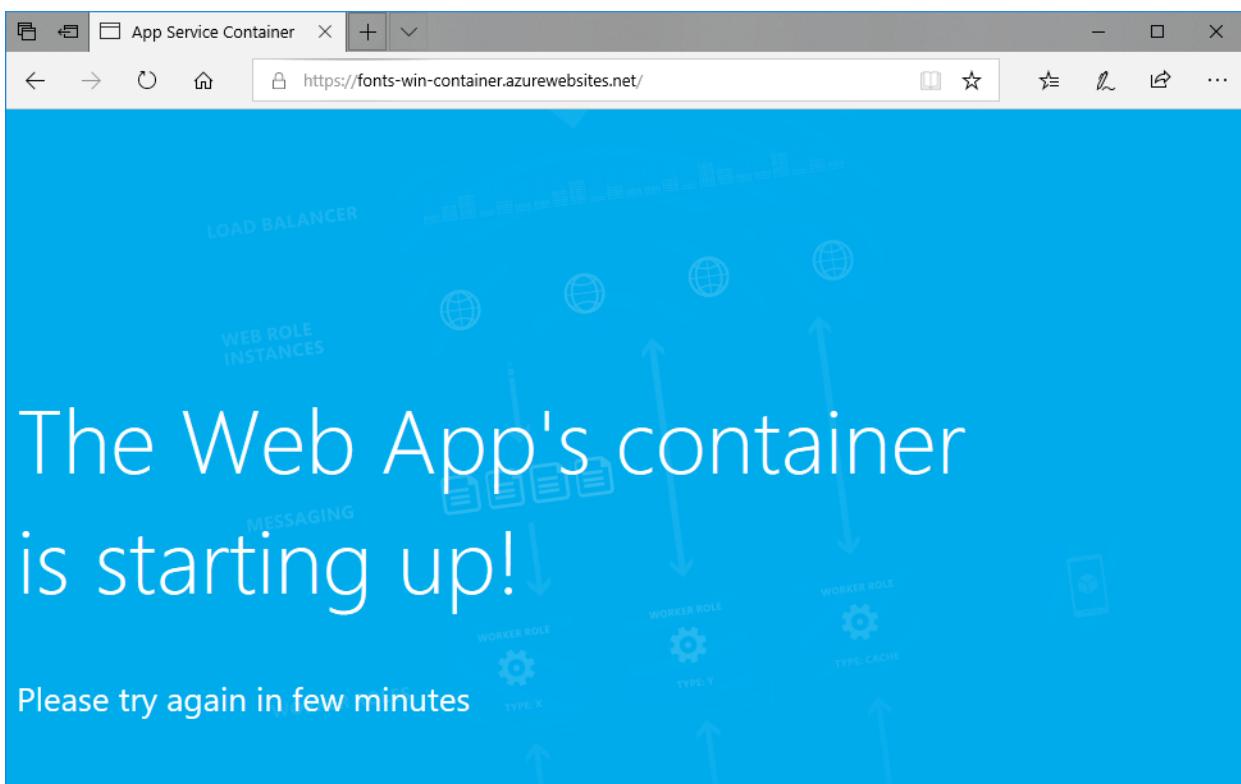
When the Azure operation is complete, a notification box is displayed.



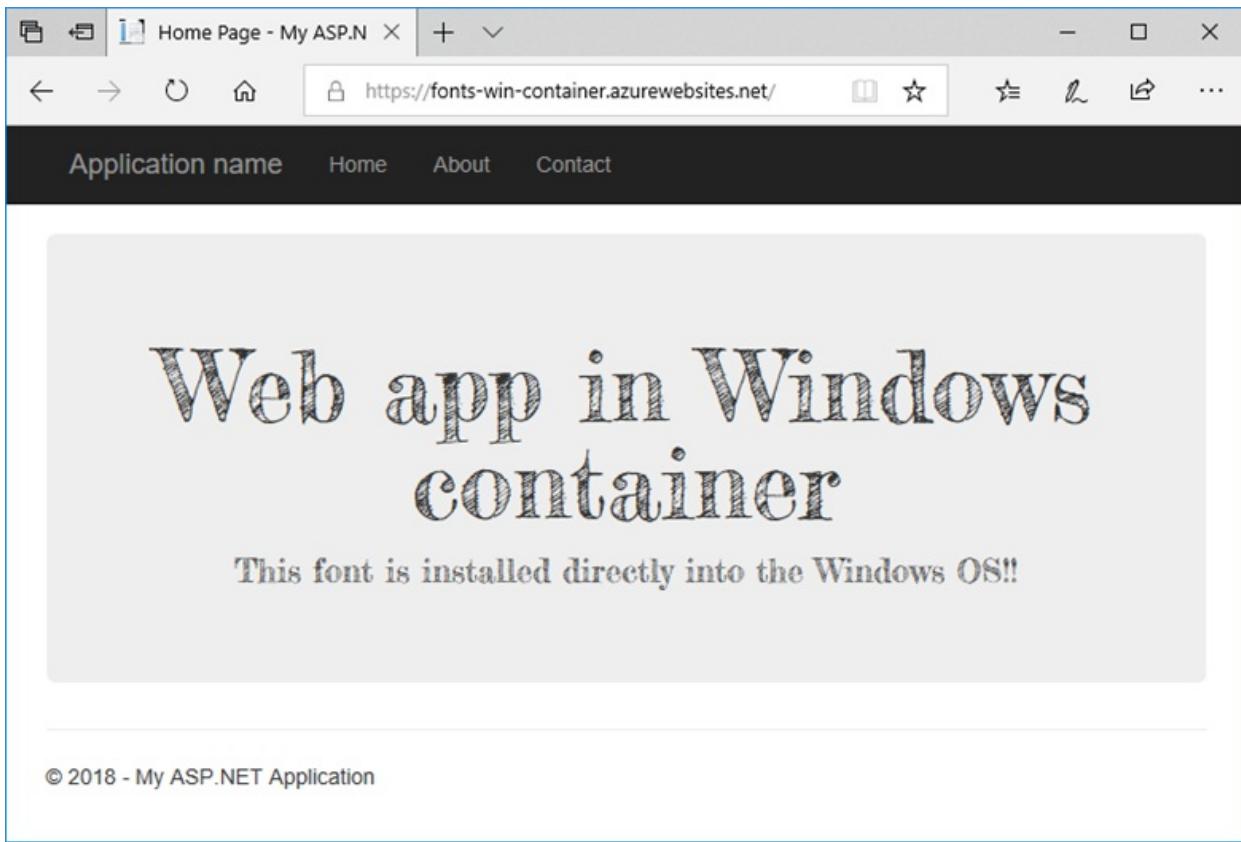
1. Click **Go to resource**.

2. In the app page, click the link under URL.

A new browser page is opened to the following page:



Wait a few minutes and try again, until you get the homepage with the beautiful font you expect:



Congratulations! You've migrated an ASP.NET application to Azure App Service in a Windows container.

## See container start-up logs

It may take some time for the Windows container to load. To see the progress, navigate to the following URL by replacing <app-name> with the name of your app.

```
https://<app-name>.scm.azurewebsites.net/api/logstream
```

The streamed logs look like this:

```
14/09/2018 23:16:19.889 INFO - Site: fonts-win-container - Creating container for image: customfontsample20180914115836.azurecr.io/customfontsample:latest.
14/09/2018 23:16:19.928 INFO - Site: fonts-win-container - Create container for image: customfontsample20180914115836.azurecr.io/customfontsample:latest succeeded. Container Id 329ecfedbe370f1d99857da7352a7633366b878607994ff1334461e44e6f5418
14/09/2018 23:17:23.405 INFO - Site: fonts-win-container - Start container succeeded. Container: 329ecfedbe370f1d99857da7352a7633366b878607994ff1334461e44e6f5418
14/09/2018 23:17:28.637 INFO - Site: fonts-win-container - Container ready
14/09/2018 23:17:28.637 INFO - Site: fonts-win-container - Configuring container
14/09/2018 23:18:03.823 INFO - Site: fonts-win-container - Container ready
14/09/2018 23:18:03.823 INFO - Site: fonts-win-container - Container start-up and configuration completed successfully
```

Azure App Service uses the Docker container technology to host both built-in images and custom images. To see a list of built-in images, run the Azure CLI command, '[az webapp list-runtimes --linux](#)'. If those images don't satisfy your needs, you can build and deploy a custom image.

In this tutorial, you learn how to:

- Push a custom Docker image to Azure Container Registry
- Deploy the custom image to App Service
- Configure environment variables

- Pull image into App Service using a managed identity
- Access diagnostic logs
- Enable CI/CD from Azure Container Registry to App Service
- Connect to the container using SSH

Completing this tutorial incurs a small charge in your Azure account for the container registry and can incur additional costs for hosting the container for longer than a month.

## Set up your initial environment

- Have an Azure account with an active subscription. [Create an account for free](#).
- Install [Docker](#), which you use to build Docker images. Installing Docker may require a computer restart.
- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This tutorial requires version 2.0.80 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

After installing Docker or running Azure Cloud Shell, open a terminal window and verify that docker is installed:

```
docker --version
```

## Clone or download the sample app

You can obtain the sample for this tutorial via git clone or download.

### Clone with git

Clone the sample repository:

```
git clone https://github.com/Azure-Samples/docker-django-webapp-linux.git --config core.autocrlf=input
```

Be sure to include the `--config core.autocrlf=input` argument to guarantee proper line endings in files that are used inside the Linux container:

Then go into that folder:

```
cd docker-django-webapp-linux
```

### Download from GitHub

Instead of using git clone, you can visit <https://github.com/Azure-Samples/docker-django-webapp-linux>, select

Clone, and then select **Download ZIP**.

Unpack the ZIP file into a folder named *docker-django-webapp-linux*.

Then open a terminal window in that *docker-django-webapp-linux* folder.

## (Optional) Examine the Docker file

The file in the sample named *Dockerfile* that describes the docker image and contains configuration instructions:

```
FROM tiangolo/uwsgi-nginx-flask:python3.6

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt --no-cache-dir
ADD . /code/

ssh
ENV SSH_PASSWD "root:Docker!"
RUN apt-get update \
 && apt-get install -y --no-install-recommends dialog \
 && apt-get update \
&& apt-get install -y --no-install-recommends openssh-server \
&& echo "$SSH_PASSWD" | chpasswd

COPY sshd_config /etc/ssh/
COPY init.sh /usr/local/bin/

RUN chmod u+x /usr/local/bin/init.sh
EXPOSE 8000 2222

#CMD ["python", "/code/manage.py", "runserver", "0.0.0.0:8000"]
ENTRYPOINT ["init.sh"]
```

- The first group of commands installs the app's requirements in the environment.
- The second group of commands create an **SSH** server for secure communication between the container and the host.
- The last line, `ENTRYPOINT ["init.sh"]`, invokes `init.sh` to start the SSH service and Python server.

## Build and test the image locally

### NOTE

Docker Hub has [quotas on the number of anonymous pulls per IP and the number of authenticated pulls per free user](#) ([see Data transfer](#)). If you notice your pulls from Docker Hub are being limited, try `docker login` if you're not already logged in.

1. Run the following command to build the image:

```
docker build --tag appsvc-tutorial-custom-image .
```

2. Test that the build works by running the Docker container locally:

```
docker run -it -p 8000:8000 appsvc-tutorial-custom-image
```

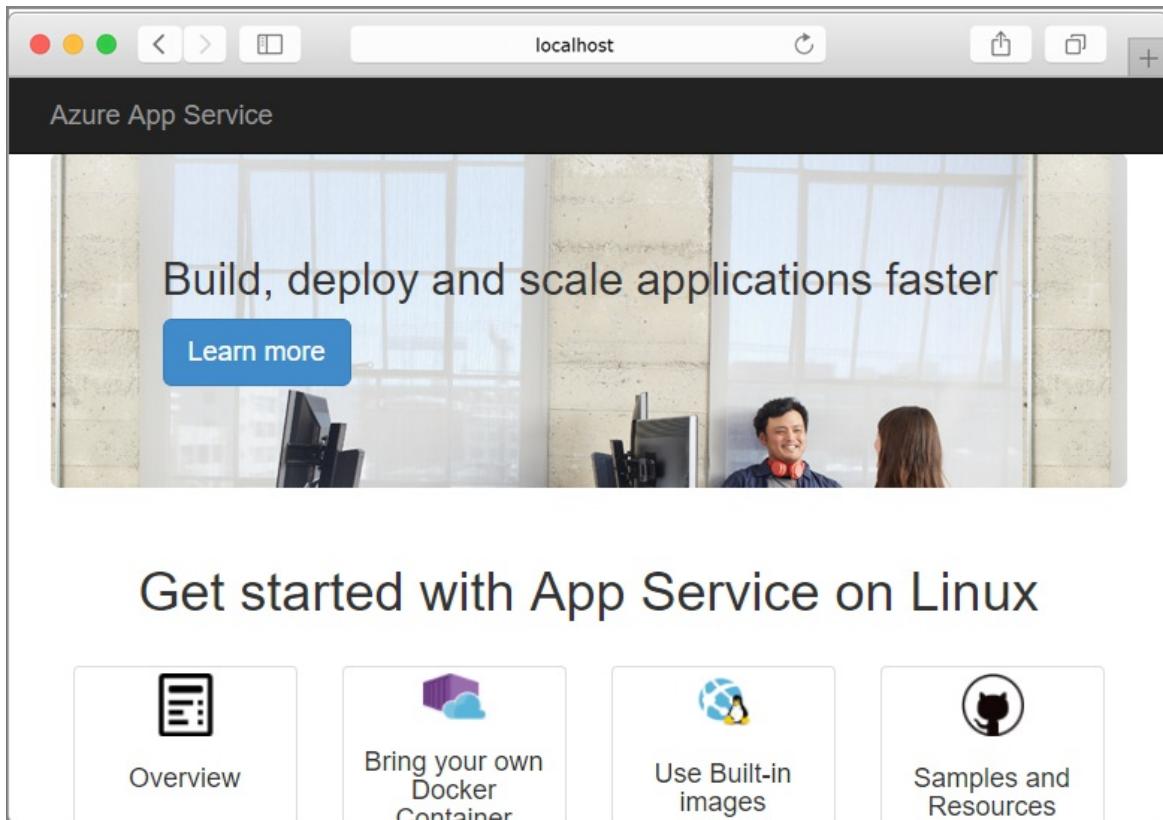
This `docker run` command specifies the port with the `-p` argument followed by the name of the image.

`-it` lets you stop it with `Ctrl+C`.

#### TIP

If you are running on Windows and see the error, `standard_init_linux.go:211: exec user process caused 'no such file or directory'`, the `init.sh` file contains CR-LF line endings instead of the expected LF endings. This error happens if you used git to clone the sample repository but omitted the `--config core.autocrlf=input` parameter. In this case, clone the repository again with the `--config` argument. You might also see the error if you edited `init.sh` and saved it with CRLF endings. In this case, save the file again with LF endings only.

3. Browse to `http://localhost:8000` to verify the web app and container are functioning correctly.



## Create a resource group

In this section and those that follow, you provision resources in Azure to which you push the image and then deploy a container to Azure App Service. You start by creating a resource group in which to collect all these resources.

Run the `az group create` command to create a resource group:

```
az group create --name myResourceGroup --location westeurope
```

You can change the `--location` value to specify a region near you.

## Push the image to Azure Container Registry

In this section, you push the image to Azure Container Registry from which App Service can deploy it.

1. Run the `az acr create` command to create an Azure Container Registry:

```
az acr create --name <registry-name> --resource-group myResourceGroup --sku Basic --admin-enabled true
```

Replace `<registry-name>` with a suitable name for your registry. The name must contain only letters and numbers and must be unique across all of Azure.

- Run the `az acr show` command to retrieve credentials for the registry:

```
az acr credential show --resource-group myResourceGroup --name <registry-name>
```

The JSON output of this command provides two passwords along with the registry's user name.

- Use the `docker login` command to sign in to the container registry:

```
docker login <registry-name>.azurecr.io --username <registry-username>
```

Replace `<registry-name>` and `<registry-username>` with values from the previous steps. When prompted, type in one of the passwords from the previous step.

You use the same registry name in all the remaining steps of this section.

- Once the login succeeds, tag your local Docker image for the registry:

```
docker tag appsvc-tutorial-custom-image <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

- Use the `docker push` command to push the image to the registry:

```
docker push <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

Uploading the image the first time might take a few minutes because it includes the base image. Subsequent uploads are typically faster.

While you're waiting, you can complete the steps in the next section to configure App Service to deploy from the registry.

- Use the `az acr repository list` command to verify that the push was successful:

```
az acr repository list -n <registry-name>
```

The output should show the name of your image.

## Configure App Service to deploy the image from the registry

To deploy a container to Azure App Service, you first create a web app on App Service, then connect the web app to the container registry. When the web app starts, App Service automatically pulls the image from the registry.

- Create an App Service plan using the `az appservice plan create` command:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --is-linux
```

An App Service plan corresponds to the virtual machine that hosts the web app. By default, the previous

command uses an inexpensive [B1 pricing tier](#) that is free for the first month. You can control the tier with the `--sku` parameter.

2. Create the web app with the `az webapp create` command:

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --deployment-container-image-name <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest
```

Replace `<app-name>` with a name for the web app, which must be unique across all of Azure. Also replace `<registry-name>` with the name of your registry from the previous section.

3. Use `az webapp config appsettings set` to set the `WEBSITES_PORT` environment variable as expected by the app code:

```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name> --settings WEBSITES_PORT=8000
```

Replace `<app-name>` with the name you used in the previous step.

For more information on this environment variable, see the [readme in the sample's GitHub repository](#).

4. Enable [the system-assigned managed identity](#) for the web app by using the `az webapp identity assign` command:

```
az webapp identity assign --resource-group myResourceGroup --name <app-name> --query principalId --output tsv
```

Replace `<app-name>` with the name you used in the previous step. The output of the command (filtered by the `--query` and `--output` arguments) is the service principal of the assigned identity, which you use shortly.

Managed identity allows you to grant permissions to the web app to access other Azure resources without needing any specific credentials.

5. Retrieve your subscription ID with the `az account show` command, which you need in the next step:

```
az account show --query id --output tsv
```

6. Grant the managed identity permission to access the container registry:

```
az role assignment create --assignee <principal-id> --scope /subscriptions/<subscription-id>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/<registry-name> -role "AcrPull"
```

Replace the following values:

- `<principal-id>` with the service principal ID from the `az webapp identity assign` command
- `<registry-name>` with the name of your container registry
- `<subscription-id>` with the subscription ID retrieved from the `az account show` command

For more information about these permissions, see [What is Azure role-based access control](#).

7. Configure your app to use the managed identity to pull from Azure Container Registry.

```
az resource update --ids /subscriptions/<subscription-id>/resourceGroups/myResourceGroup/providers/Microsoft.Web/sites/<app-name>/config/web --set properties.acrUseManagedIdentityCreds=True
```

Replace the following values:

- <subscription-id> with the subscription ID retrieved from the `az account show` command.
- <app-name> with the name of your web app.

#### TIP

If your app uses a [user-assigned managed identity](#), set an additional `AcrUserManagedIdentityID` property to specify its client ID:

```
clientId=$(az identity show --resource-group <group-name> --name <identity-name> --query clientId --output tsv)
az resource update --ids /subscriptions/<subscription-id>/resourceGroups/myResourceGroup/providers/Microsoft.Web/sites/<app-name>/config/web --set properties.AcrUserManagedIdentityID=$clientId
```

## Deploy the image and test the app

You can complete these steps once the image is pushed to the container registry and the App Service is fully provisioned.

1. Use the `az webapp config container set` command to specify the container registry and the image to deploy for the web app:

```
az webapp config container set --name <app-name> --resource-group myResourceGroup --docker-custom-image-name <registry-name>.azurecr.io/appsvc-tutorial-custom-image:latest --docker-registry-server-url https://<registry-name>.azurecr.io
```

Replace <app-name> with the name of your web app and replace <registry-name> in two places with the name of your registry.

- When using a registry other than Docker Hub (as this example shows), `--docker-registry-server-url` must be formatted as `https://` followed by the fully qualified domain name of the registry.
- The message, "No credential was provided to access Azure Container Registry. Trying to look up..." tells you that Azure is using the app's managed identity to authenticate with the container registry rather than asking for a username and password.
- If you encounter the error, "AttributeError: 'NoneType' object has no attribute 'reserved'", make sure your <app-name> is correct.

#### TIP

You can retrieve the web app's container settings at any time with the command

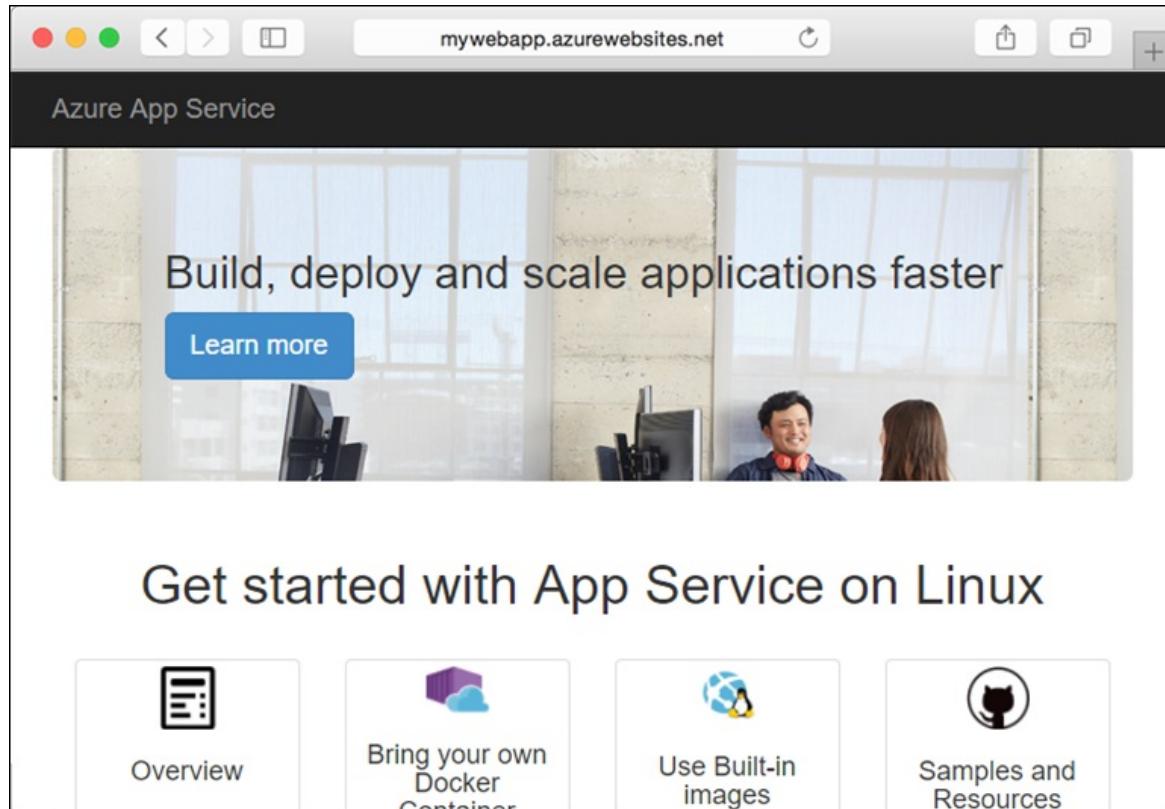
```
az webapp config container show --name <app-name> --resource-group myResourceGroup
```

. The image is specified in the property `DOCKER_CUSTOM_IMAGE_NAME`. When the web app is deployed through Azure DevOps or Azure Resource Manager templates, the image can also appear in a property named `LinuxFxVersion`. Both properties serve the same purpose. If both are present in the web app's configuration, `LinuxFxVersion` takes precedence.

2. Once the `az webapp config container set` command completes, the web app should be running in the

container on App Service.

To test the app, browse to `https://<app-name>.azurewebsites.net`, replacing `<app-name>` with the name of your web app. On first access, it may take some time for the app to respond because App Service must pull the entire image from the registry. If the browser times out, just refresh the page. Once the initial image is pulled, subsequent tests will run much faster.



## Access diagnostic logs

While you're waiting for App Service to pull in the image, it's helpful to see exactly what App Service is doing by streaming the container logs to your terminal.

1. Turn on container logging:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

2. Enable the log stream:

```
az webapp log tail --name <app-name> --resource-group myResourceGroup
```

If you don't see console logs immediately, check again in 30 seconds.

You can also inspect the log files from the browser at

`https://<app-name>.scm.azurewebsites.net/api/logs/docker`.

3. To stop log streaming at any time, type `Ctrl+C`.

## Configure continuous deployment

Your App Service app now can pull the container image securely from your private container registry. However, it doesn't know when that image is updated in your registry. Each time you push the updated image to the

registry, you must manually trigger an image pull by restarting the App Service app. In this step, you enable CI/CD, so that App Service gets notified of a new image and triggers a pull automatically.

1. Enable CI/CD in App Service.

```
az webapp deployment container config --enable-cd true --name <app-name> --resource-group myResourceGroup --query CI_CD_URL --output tsv
```

`CI_CD_URL` is a URL that App Service generates for you. Your registry should use this URL to notify App Service that an image push occurred. It doesn't actually create the webhook for you.

2. Create a webhook in your container registry using the `CI_CD_URL` you got from the last step.

```
az acr webhook create --name appserviceCD --registry <registry-name> --uri '<ci-cd-url>' --actions push --scope appsvc-tutorial-custom-image:latest
```

3. To test if your webhook is configured properly, ping the webhook and see if you get a 200 OK response.

```
eventId=$(az acr webhook ping --name appserviceCD --registry <registry-name> --query id --output tsv)
az acr webhook list-events --name appserviceCD --registry <registry-name> --query "[? id=='$eventId'].eventResponseMessage"
```

**TIP**

To see all information about all webhook events, remove the `--query` parameter.

If you're streaming the container log, you should see the message after the webhook ping:

`Starting container for site`, because the webhook triggers the app to restart. Since you haven't made anything updates to the image, there's nothing new for App Service to pull.

## Modify the app code and redeploy

In this section, you make a change to the web app code, rebuild the image, and then push it to your container registry. App Service then automatically pulls the updated image from the registry to update the running web app.

1. In your local `docker-django-webapp-linux` folder, open the file `app/templates/app/index.html`.

2. Change the first HTML element to match the following code.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
 <div class="container">
 <div class="navbar-header">
 Azure App Service - Updated Here!
 </div>
 </div>
</nav>
```

3. Save your changes.

4. Change to the `docker-django-webapp-linux` folder and rebuild the image:

```
docker build --tag appsvc-tutorial-custom-image .
```

5. Update the version number in the image's tag to v1.0.1:

```
docker tag appsvc-tutorial-custom-image <registry-name>.azurecr.io/appsvc-tutorial-custom-image:v1.0.1
```

Replace `<registry-name>` with the name of your registry.

- Push the image to the registry:

```
docker push <registry-name>.azurecr.io/appsvc-tutorial-custom-image:v1.0.1
```

- Once the image push is complete, the webhook notifies App Service about the push, and App Service tries to pull in the updated image. Wait a few minutes, and then verify that the update has been deployed by browsing to <https://<app-name>.azurewebsites.net>.

## Connect to the container using SSH

SSH enables secure communication between a container and a client. To enable SSH connection to your container, your custom image must be configured for it. Once the container is running, you can open an SSH connection.

### Configure the container for SSH

The sample app used in this tutorial already has the necessary configuration in the *Dockerfile*, which installs the SSH server and also sets the login credentials. This section is informational only. To connect to the container, skip to the next section

```
ENV SSH_PASSWD "root:Docker!"
RUN apt-get update \
 && apt-get install -y --no-install-recommends dialog \
 && apt-get update \
 && apt-get install -y --no-install-recommends openssh-server \
 && echo "$SSH_PASSWD" | chpasswd
```

#### NOTE

This configuration doesn't allow external connections to the container. SSH is available only through the Kudu/SCM Site. The Kudu/SCM site is authenticated with your Azure account. `root:Docker!` should not be altered SSH. SCM/KUDU will use your Azure Portal credentials. Changing this value will result in an error when using SSH.

The *Dockerfile* also copies the *sshd\_config* file to the */etc/ssh/* folder and exposes port 2222 on the container:

```
COPY sshd_config /etc/ssh/

...

EXPOSE 8000 2222
```

Port 2222 is an internal port accessible only by containers within the bridge network of a private virtual network.

Finally, the entry script, *init.sh*, starts the SSH server.

```
#!/bin/bash
service ssh start
```

## Open SSH connection to container

1. Browse to `https://<app-name>.scm.azurewebsites.net/webssh/host` and sign in with your Azure account. Replace `<app-name>` with the name of your web app.
2. Once signed in, you're redirected to an informational page for the web app. Select **SSH** at the top of the page to open the shell and use commands.  
For example, you can examine the processes running within it using the `top` command.

## Clean up resources

The resources you created in this article may incur ongoing costs. To clean up the resources, you need only delete the resource group that contains them:

```
az group delete --name myResourceGroup
```

## Next steps

What you learned:

- Deploy a custom image to a private container registry
- Deploy and the custom image in App Service
- Update and redeploy the image
- Access diagnostic logs
- Connect to the container using SSH
- Push a custom Docker image to Azure Container Registry
- Deploy the custom image to App Service
- Configure environment variables
- Pull image into App Service using a managed identity
- Access diagnostic logs
- Enable CI/CD from Azure Container Registry to App Service
- Connect to the container using SSH

In the next tutorial, you learn how to map a custom DNS name to your app.

[Tutorial: Map custom DNS name to your app](#)

Or, check out other resources:

[Configure custom container](#)

[Tutorial: Multi-container WordPress app](#)

# Tutorial: Create a multi-container (preview) app in Web App for Containers

11/2/2021 • 11 minutes to read • [Edit Online](#)

## NOTE

Multi-container is in preview.

[Web App for Containers](#) provides a flexible way to use Docker images. In this tutorial, you'll learn how to create a multi-container app using WordPress and MySQL. You'll complete this tutorial in Cloud Shell, but you can also run these commands locally with the [Azure CLI](#) command-line tool (2.0.32 or later).

In this tutorial, you learn how to:

- Convert a Docker Compose configuration to work with Web App for Containers
- Deploy a multi-container app to Azure
- Add application settings
- Use persistent storage for your containers
- Connect to Azure Database for MySQL
- Troubleshoot errors

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial, you need experience with [Docker Compose](#).

## Download the sample

For this tutorial, you use the compose file from [Docker](#), but you'll modify it to include Azure Database for MySQL, persistent storage, and Redis. The configuration file can be found at [Azure Samples](#). For supported configuration options, see [Docker Compose options](#).

```
version: '3.3'

services:
 db:
 image: mysql:5.7
 volumes:
 - db_data:/var/lib/mysql
 restart: always
 environment:
 MYSQL_ROOT_PASSWORD: somewordpress
 MYSQL_DATABASE: wordpress
 MYSQL_USER: wordpress
 MYSQL_PASSWORD: wordpress

 wordpress:
 depends_on:
 - db
 image: wordpress:latest
 ports:
 - "8000:80"
 restart: always
 environment:
 WORDPRESS_DB_HOST: db:3306
 WORDPRESS_DB_USER: wordpress
 WORDPRESS_DB_PASSWORD: wordpress
 volumes:
 db_data:
```

In Cloud Shell, create a tutorial directory and then change to it.

```
mkdir tutorial
cd tutorial
```

Next, run the following command to clone the sample app repository to your tutorial directory. Then change to the `multicontainerwordpress` directory.

```
git clone https://github.com/Azure-Samples/multicontainerwordpress
cd multicontainerwordpress
```

## Create a resource group

A **resource group** is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one simple step later.

In Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named *myResourceGroup* in the *South Central US* location. To see all supported locations for App Service on Linux in Standard tier, run the `az appservice list-locations --sku S1 --linux-workers-enabled` command.

```
az group create --name myResourceGroup --location "South Central US"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

# Create an Azure App Service plan

In Cloud Shell, create an App Service plan in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Standard** pricing tier (`--sku S1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku S1 --is-linux
```

When the App Service plan has been created, Cloud Shell shows information similar to the following example:

```
{
 "adminSiteName": null,
 "appServicePlanName": "myAppServicePlan",
 "geoRegion": "South Central US",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-
0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "linux",
 "location": "South Central US",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

## Docker Compose with WordPress and MySQL containers

### Create a Docker Compose app

In your Cloud Shell, create a multi-container [web app](#) in the `myAppServicePlan` App Service plan with the `az webapp create` command. Don't forget to replace `<app-name>` with a unique app name.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --
multicontainer-config-type compose --multicontainer-config-file docker-compose-wordpress.yml
```

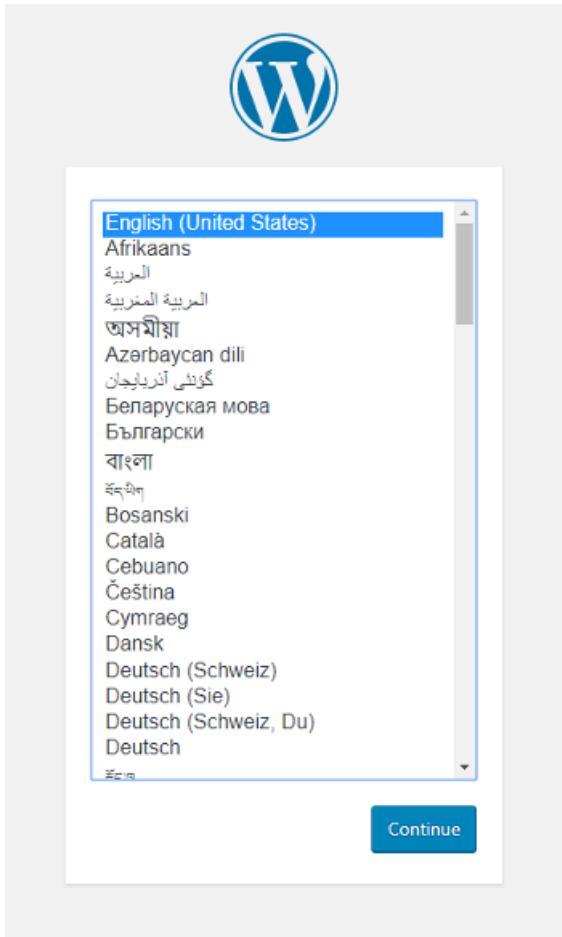
When the web app has been created, Cloud Shell shows output similar to the following example:

```
{
 "additionalProperties": {},
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

## Browse to the app

Browse to the deployed app at (<http://<app-name>.azurewebsites.net>). The app may take a few minutes to load. If you receive an error, allow a few more minutes then refresh the browser. If you're having trouble and would

like to troubleshoot, review [container logs](#).



**Congratulations**, you've created a multi-container app in Web App for Containers. Next you'll configure your app to use Azure Database for MySQL. Don't install WordPress at this time.

## Connect to production database

It's not recommended to use database containers in a production environment. The local containers aren't scalable. Instead, you'll use Azure Database for MySQL which can be scaled.

### Create an Azure Database for MySQL server

Create an Azure Database for MySQL server with the `az mysql server create` command.

In the following command, substitute your MySQL server name where you see the `<mysql-server-name>` placeholder (valid characters are `a-z`, `0-9`, and `-`). This name is part of the MySQL server's hostname (`<mysql-server-name>.database.windows.net`), it needs to be globally unique.

```
az mysql server create --resource-group myResourceGroup --name <mysql-server-name> --location "South Central US" --admin-user adminuser --admin-password My5up3rStr0ngPaSw0rd! --sku-name B_Gen5_1 --version 5.7
```

Creating the server may take a few minutes to complete. When the MySQL server is created, Cloud Shell shows information similar to the following example:

```
{
 "administratorLogin": "adminuser",
 "administratorLoginPassword": null,
 "fullyQualifiedDomainName": "<mysql-server-name>.database.windows.net",
 "id": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/<mysql-
server-name>",
 "location": "southcentralus",
 "name": "<mysql-server-name>",
 "resourceGroup": "myResourceGroup",
 ...
}
```

## Configure server firewall

Create a firewall rule for your MySQL server to allow client connections by using the

`az mysql server firewall-rule create` command. When both starting IP and end IP are set to 0.0.0.0, the firewall is only opened for other Azure resources.

```
az mysql server firewall-rule create --name allAzureIPs --server <mysql-server-name> --resource-group
myResourceGroup --start-ip-address 0.0.0.0 --end-ip-address 0.0.0.0
```

### TIP

You can be even more restrictive in your firewall rule by [using only the outbound IP addresses your app uses](#).

## Create the WordPress database

```
az mysql db create --resource-group myResourceGroup --server-name <mysql-server-name> --name wordpress
```

When the database has been created, Cloud Shell shows information similar to the following example:

```
{
 "additionalProperties": {},
 "charset": "latin1",
 "collation": "latin1_swedish_ci",
 "id": "/subscriptions/12db1644-4b12-4cab-ba54-
8ba2f2822c1f/resourceGroups/myResourceGroup/providers/Microsoft.DBforMySQL/servers/<mysql-
server-name>/databases/wordpress",
 "name": "wordpress",
 "resourceGroup": "myResourceGroup",
 "type": "Microsoft.DBforMySQL/servers/databases"
}
```

## Configure database variables in WordPress

To connect the WordPress app to this new MySQL server, you'll configure a few WordPress-specific environment variables, including the SSL CA path defined by `MYSQL_SSL_CA`. The [Baltimore CyberTrust Root](#) from [DigiCert](#) is provided in the [custom image](#) below.

To make these changes, use the `az webapp config appsettings set` command in Cloud Shell. App settings are case-sensitive and space-separated.

```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name> --settings
WORDPRESS_DB_HOST=<mysql-server-name>.mysql.database.azure.com WORDPRESS_DB_USER="adminuser@<mysql-server-
name>" WORDPRESS_DB_PASSWORD="My5up3rStr0ngPaSw0rd!" WORDPRESS_DB_NAME="wordpress"
MYSQL_SSL_CA="BaltimoreCyberTrustroot.crt.pem"
```

When the app setting has been created, Cloud Shell shows information similar to the following example:

```
[
 {
 "name": "WORDPRESS_DB_HOST",
 "slotSetting": false,
 "value": "<mysql-server-name>.mysql.database.azure.com"
,
 {
 "name": "WORDPRESS_DB_USER",
 "slotSetting": false,
 "value": "adminuser@<mysql-server-name>"
,
 {
 "name": "WORDPRESS_DB_NAME",
 "slotSetting": false,
 "value": "wordpress"
,
 {
 "name": "WORDPRESS_DB_PASSWORD",
 "slotSetting": false,
 "value": "My5up3rStr0ngPaSw0rd!"
,
 {
 "name": "MYSQL_SSL_CA",
 "slotSetting": false,
 "value": "BaltimoreCyberTrustroot.crt.pem"
 }
]
```

For more information on environment variables, see [Configure environment variables](#).

### Use a custom image for MySQL TLS/SSL and other configurations

By default, TLS/SSL is used by Azure Database for MySQL. WordPress requires additional configuration to use TLS/SSL with MySQL. The WordPress 'official image' doesn't provide the additional configuration, but a [custom image](#) has been prepared for your convenience. In practice, you would add desired changes to your own image.

The custom image is based on the 'official image' of [WordPress from Docker Hub](#). The following changes have been made in this custom image for Azure Database for MySQL:

- Adds Baltimore Cyber Trust Root Certificate file for SSL to MySQL.
- Uses App Setting for MySQL SSL Certificate Authority certificate in WordPress wp-config.php.
- Adds WordPress define for MYSQL\_CLIENT\_FLAGS needed for MySQL SSL.

The following changes have been made for Redis (to be used in a later section):

- Adds PHP extension for Redis v4.0.2.
- Adds unzip needed for file extraction.
- Adds Redis Object Cache 1.3.8 WordPress plugin.
- Uses App Setting for Redis host name in WordPress wp-config.php.

To use the custom image, you'll update your docker-compose-wordpress.yml file. In Cloud Shell, type

`nano docker-compose-wordpress.yml` to open the nano text editor. Change the `image: wordpress` to use `image: mcr.microsoft.com/azuredocs/multicontainerwordpress`. You no longer need the database container. Remove the `db`, `environment`, `depends_on`, and `volumes` section from the configuration file. Your file should look like the following code:

```
version: '3.3'

services:
 wordpress:
 image: mcr.microsoft.com/azuredocs/multicontainerwordpress
 ports:
 - "8000:80"
 restart: always
```

Save your changes and exit nano. Use the command `^O` to save and `^X` to exit.

### Update app with new configuration

In Cloud Shell, reconfigure your multi-container web app with the `az webapp config container set` command. Don't forget to replace `<app-name>` with the name of the web app you created earlier.

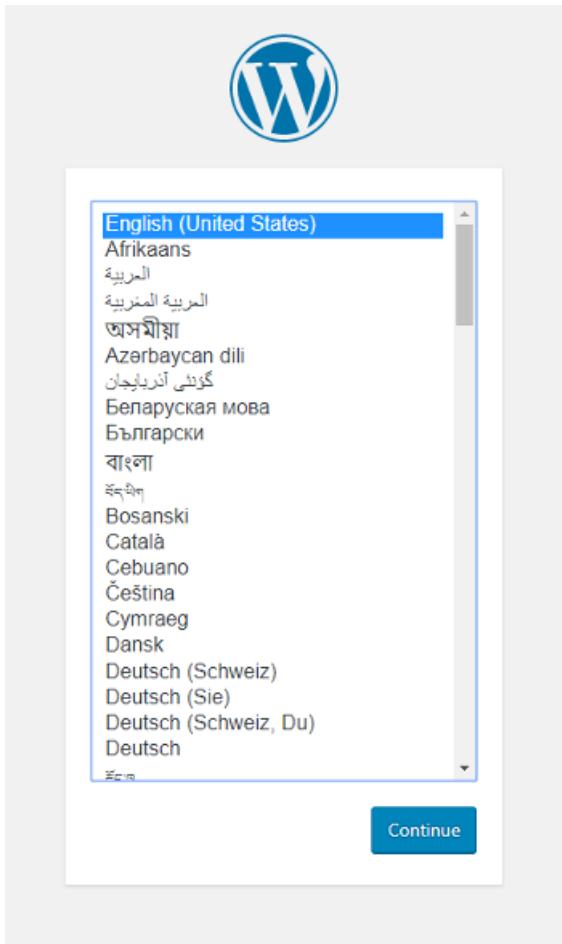
```
az webapp config container set --resource-group myResourceGroup --name <app-name> --multicontainer-config-type compose --multicontainer-config-file docker-compose-wordpress.yml
```

When the app has been reconfigured, Cloud Shell shows information similar to the following example:

```
[
 {
 "name": "DOCKER_CUSTOM_IMAGE_NAME",
 "value":
 "COMPOSE|dmVyc2lvbjogJzMzMycKCNlcnZpY2Vz0gogICB3b3JkcHJlc3M6CiAgICAgaW1hZ2U6IG1zYW5nYXB1L3dvcmR
 wcmVzcwogICAgIHBvcnRz0gogICAgICAgLSAiODAwMDo4MCIKICAgICByZXN0YXJ0iBhbHdheXM="
 }
]
```

### Browse to the app

Browse to the deployed app at (`http://<app-name>.azurewebsites.net`). The app is now using Azure Database for MySQL.



## Add persistent storage

Your multi-container is now running in Web App for Containers. However, if you install WordPress now and restart your app later, you'll find that your WordPress installation is gone. This happens because your Docker Compose configuration currently points to a storage location inside your container. The files installed into your container don't persist beyond app restart. In this section, you'll [add persistent storage](#) to your WordPress container.

### Configure environment variables

To use persistent storage, you'll enable this setting within App Service. To make this change, use the [az webapp config appsettings set](#) command in Cloud Shell. App settings are case-sensitive and space-separated.

```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name> --settings
WEBSITES_ENABLE_APP_SERVICE_STORAGE=TRUE
```

When the app setting has been created, Cloud Shell shows information similar to the following example:

```
[
 < JSON data removed for brevity. >
 {
 "name": "WORDPRESS_DB_NAME",
 "slotSetting": false,
 "value": "wordpress"
 },
 {
 "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",
 "slotSetting": false,
 "value": "TRUE"
 }
]
```

## Modify configuration file

In the Cloud Shell, type `nano docker-compose-wordpress.yml` to open the nano text editor.

The `volumes` option maps the file system to a directory within the container.  `${WEBAPP_STORAGE_HOME}` is an environment variable in App Service that is mapped to persistent storage for your app. You'll use this environment variable in the volumes option so that the WordPress files are installed into persistent storage instead of the container. Make the following modifications to the file:

In the `wordpress` section, add a `volumes` option so it looks like the following code:

```
version: '3.3'

services:
 wordpress:
 image: mcr.microsoft.com/azuredocs/multicontainerwordpress
 volumes:
 - ${WEBAPP_STORAGE_HOME}/site/wwwroot:/var/www/html
 ports:
 - "8000:80"
 restart: always
```

## Update app with new configuration

In Cloud Shell, reconfigure your multi-container web app with the [az webapp config container set](#) command. Don't forget to replace `<app-name>` with a unique app name.

```
az webapp config container set --resource-group myResourceGroup --name <app-name> --multicontainer-config-type compose --multicontainer-config-file docker-compose-wordpress.yml
```

After your command runs, it shows output similar to the following example:

```
[
 {
 "name": "WEBSITES_ENABLE_APP_SERVICE_STORAGE",
 "slotSetting": false,
 "value": "TRUE"
 },
 {
 "name": "DOCKER_CUSTOM_IMAGE_NAME",
 "value":
 "COMPOSE | dmVyc2lvbjogJzMuMycKCnNlcnPZpY2Vz0gogICBteXNxbDoKICAgICBpbWFnZTogbX1zcWw6NS43CiAgICAgdm9
 sdW1lcz0KICAgICAgIC0gZGJfZGF0YTovdmFyL2xpYi9teXNxbAogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25
 tZW500gogICAgICAgTV1TUUxfUk9PVF9QQVNTV09SRDogZXhhbXBsZXhC3MKCiAgIHdvcmRwcmVzczoKICAgICBkZXBlbmR
 zX29u0gogICAgICAgLSBteXNxbAogICAgIGltYWd1OiB3b3JkcHJlc3M6bGF0ZXN0CiAgICAgcG9ydHM6CiAgICAgICAtICI
 4MDAwOjgwIgogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW500gogICAgICAgV09SRFBSRVNTX0RCX1BBU1N
 XT1JE0iBleGFtcGxlcGFzwp2b2x1bWVz0gogICAgZGJfZGF0YTo="
 }
]
```

## Browse to the app

Browse to the deployed app at (<http://<app-name>.azurewebsites.net>).

The WordPress container is now using Azure Database for MySQL and persistent storage.

## Add Redis container

The WordPress 'official image' does not include the dependencies for Redis. These dependencies and additional configuration needed to use Redis with WordPress have been prepared for you in this [custom image](#). In practice, you would add desired changes to your own image.

The custom image is based on the 'official image' of [WordPress from Docker Hub](#). The following changes have been made in this custom image for Redis:

- Adds PHP extension for Redis v4.0.2.
- Adds unzip needed for file extraction.
- Adds Redis Object Cache 1.3.8 WordPress plugin.
- Uses App Setting for Redis host name in WordPress wp-config.php.

Add the redis container to the bottom of the configuration file so it looks like the following example:

```
version: '3.3'

services:
 wordpress:
 image: mcr.microsoft.com/azuredocs/multicontainerwordpress
 ports:
 - "8000:80"
 restart: always

 redis:
 image: mcr.microsoft.com/oss/bitnami/redis:6.0.8
 environment:
 - ALLOW_EMPTY_PASSWORD=yes
 restart: always
```

## Configure environment variables

To use Redis, you'll enable this setting, `WP_REDIS_HOST`, within App Service. This is a *required setting* for WordPress to communicate with the Redis host. To make this change, use the [az webapp config appsettings set](#) command in Cloud Shell. App settings are case-sensitive and space-separated.

```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name> --settings
WP_REDIS_HOST="redis"
```

When the app setting has been created, Cloud Shell shows information similar to the following example:

```
[
 < JSON data removed for brevity. >
 {
 "name": "WORDPRESS_DB_USER",
 "slotSetting": false,
 "value": "adminuser@<mysql-server-name>"
 },
 {
 "name": "WP_REDIS_HOST",
 "slotSetting": false,
 "value": "redis"
 }
]
```

## Update app with new configuration

In Cloud Shell, reconfigure your multi-container [web app](#) with the `az webapp config container set` command.

Don't forget to replace `<app-name>` with a unique app name.

```
az webapp config container set --resource-group myResourceGroup --name <app-name> --multicontainer-config-type compose --multicontainer-config-file compose-wordpress.yml
```

After your command runs, it shows output similar to the following example:

```
[
 {
 "name": "DOCKER_CUSTOM_IMAGE_NAME",
 "value": "
"COMPOSE|dmVyc2lvbjogJzMzMycKCnNlcnPzY2VzOgogICBteXNxbDoKICAgICBpbWFnZTogbX1zcWw6NS43CiAgICAgdm9
sdW1lczokICAgICAgIC0gZGJfZGF0YTovdmFyL2xpYi9teXNxbAogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25
tZW50ogogICAgICAgICAgTV1TUUxfUk9PVF9QQVNTV09SRDogZXhhbXBc3MKCiAgIHdvcmRwcmVzczoKICAgICBkZXBlbmR
zX29u0gogICAgICAgLSBteXNxbAogICAgIGltYWdl0iB3b3JkcHJlc3M6bGF0ZXN0CiAgICAgcG9ydHM6CiAgICAgICAtICI
4MDAwOjgwIgogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW50ogogICAgICAgV09SRFBSRVNTX0RCX1BBU1N
XT1JE0iBleGFtcGxlcGFzcp2b2x1bWVzOgogICAgZGJfZGF0YTo="
 }
]
```

## Browse to the app

Browse to the deployed app at (<http://<app-name>.azurewebsites.net>).

Complete the steps and install WordPress.

## Connect WordPress to Redis

Sign in to WordPress admin. In the left navigation, select **Plugins**, and then select **Installed Plugins**.

The screenshot shows the WordPress dashboard at [multi-app1.azurewebsites.net/wp-admin/index.php](https://multi-app1.azurewebsites.net/wp-admin/index.php). The left sidebar has a red box around the 'Plugins' item, which is currently selected. A dropdown menu for 'Plugins' is open, showing 'Installed Plugins' (selected), 'Add New', and 'Editor'. The main content area displays the 'Welcome to WordPress!' message and a 'Get Started' section with a 'Customize Your Site' button. Below that, it says 'or, change your theme completely'. At the bottom, it shows '1 Comment' and 'WordPress 4.9.5 running Twenty Seventeen theme.'

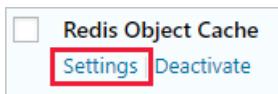
Show all plugins here

In the plugins page, find **Redis Object Cache** and click **Activate**.

The screenshot shows the 'Plugins' page in the WordPress admin. The 'Redis Object Cache' plugin is listed in the table. The 'Activate' link for this plugin is highlighted with a red box. The table columns are 'Plugin' and 'Description'. Other plugins listed include 'Akismet Anti-Spam' and 'Hello Dolly'.

Plugin	Description
Akismet Anti-Spam <a href="#">Activate</a>   <a href="#">Delete</a>	Used by millions, Akismet is quite possibly the best protected even while you sleep. To get started: activate and set up your API key. Version 4.0.3   By <a href="#">Automatic</a>   <a href="#">View details</a>
Hello Dolly <a href="#">Activate</a>   <a href="#">Delete</a>	This is not just a plugin, it symbolizes the hope and most famously by Louis Armstrong: Hello, Dolly. Will appear right of your admin screen on every page. Version 1.7   By <a href="#">Matt Mullenweg</a>   <a href="#">View details</a>
Redis Object Cache <a href="#">Activate</a>   <a href="#">Delete</a>	A persistent object cache backend powered by Redis. Version 1.3.8   By <a href="#">Till Krüss</a>   <a href="#">View details</a>
Plugin	Description

Click on **Settings**.



Click the **Enable Object Cache** button.

Redis Object Cache

Overview

Status: **Disabled**

**Enable Object Cache**

WordPress connects to the Redis server. The connection **status** appears on the same page.

Redis Object Cache

Object cache enabled. ×

Overview

Status: **Connected**

Client: **PhpRedis (v4.0.2)**

**Flush Cache** **Disable Object Cache**

Servers

Alias	Protocol	Host	Port	Database	Password
Master	TCP	redis	6379	0	No
Alias	Protocol	Host	Port	Database	Password

[Show Diagnostics](#)

**Congratulations**, you've connected WordPress to Redis. The production-ready app is now using **Azure Database for MySQL, persistent storage, and Redis**. You can now scale out your App Service Plan to multiple instances.

## Find Docker Container logs

If you run into issues using multiple containers, you can access the container logs by browsing to:

`https://<app-name>.scm.azurewebsites.net/api/logs/docker`.

You'll see output similar to the following example:

```
[
 {
 "machineName": "RD00XYZYZE567A",
 "lastUpdated": "2018-05-10T04:11:45Z",
 "size": 25125,
 "href": "https://<app-name>.scm.azurewebsites.net/api/vfs/LogFiles/2018_05_10_RD00XYZYZE567A_docker.log",
 "path": "/home/LogFiles/2018_05_10_RD00XYZYZE567A_docker.log"
 }
]
```

You see a log for each container and an additional log for the parent process. Copy the respective `href` value into the browser to view the log.

## Clean up deployment

After the sample script has been run, the following command can be used to remove the resource group and all resources associated with it.

```
az group delete --name myResourceGroup
```

## Next steps

In this tutorial, you learned how to:

- Convert a Docker Compose configuration to work with Web App for Containers
- Deploy a multi-container app to Azure
- Add application settings
- Use persistent storage for your containers
- Connect to Azure Database for MySQL
- Troubleshoot errors

Advance to the next tutorial to learn how to map a custom DNS name to your app.

[Tutorial: Map custom DNS name to your app](#)

Or, check out other resources:

- [Configure custom container](#)
- [Environment variables and app settings reference](#)

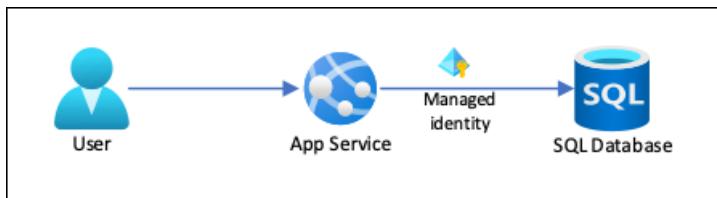
# Tutorial: Secure Azure SQL Database connection from App Service using a managed identity

11/2/2021 • 12 minutes to read • [Edit Online](#)

App Service provides a highly scalable, self-patching web hosting service in Azure. It also provides a [managed identity](#) for your app, which is a turn-key solution for securing access to [Azure SQL Database](#) and other Azure services. Managed identities in App Service make your app more secure by eliminating secrets from your app, such as credentials in the connection strings. In this tutorial, you will add managed identity to the sample web app you built in one of the following tutorials:

- [Tutorial: Build an ASP.NET app in Azure with Azure SQL Database](#)
- [Tutorial: Build an ASP.NET Core and Azure SQL Database app in Azure App Service](#)

When you're finished, your sample app will connect to SQL Database securely without the need of username and passwords.



## NOTE

The steps covered in this tutorial support the following versions:

- .NET Framework 4.7.2 and above
- .NET Core 2.2 and above

What you will learn:

- Enable managed identities
- Grant SQL Database access to the managed identity
- Configure Entity Framework to use Azure AD authentication with SQL Database
- Connect to SQL Database from Visual Studio using Azure AD authentication

## NOTE

Azure AD authentication is *different* from [Integrated Windows authentication](#) in on-premises Active Directory (AD DS).

AD DS and Azure AD use completely different authentication protocols. For more information, see [Azure AD Domain Services documentation](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

This article continues where you left off in [Tutorial: Build an ASP.NET app in Azure with SQL Database](#) or [Tutorial: Build an ASP.NET Core and SQL Database app in Azure App Service](#). If you haven't already, follow one of the two tutorials first. Alternatively, you can adapt the steps for your own .NET app with SQL Database.

To debug your app using SQL Database as the back end, make sure that you've allowed client connection from your computer. If not, add the client IP by following the steps at [Manage server-level IP firewall rules using the Azure portal](#).

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

## Grant database access to Azure AD user

First enable Azure AD authentication to SQL Database by assigning an Azure AD user as the Active Directory admin of the server. This user is different from the Microsoft account you used to sign up for your Azure subscription. It must be a user that you created, imported, synced, or invited into Azure AD. For more information on allowed Azure AD users, see [Azure AD features and limitations in SQL Database](#).

1. If your Azure AD tenant doesn't have a user yet, create one by following the steps at [Add or delete users using Azure Active Directory](#).
2. Find the object ID of the Azure AD user using the `az ad user list` and replace `<user-principal-name>`.  
The result is saved to a variable.

```
azureaduser=$(az ad user list --filter "userPrincipalName eq '<user-principal-name>'" --query
[].objectId --output tsv)
```

### TIP

To see the list of all user principal names in Azure AD, run `az ad user list --query [].userPrincipalName`.

3. Add this Azure AD user as an Active Directory admin using `az sql server ad-admin create` command in the Cloud Shell. In the following command, replace `<server-name>` with the server name (without the `.database.windows.net` suffix).

```
az sql server ad-admin create --resource-group myResourceGroup --server-name <server-name> --display-
name ADMIN --object-id $azureaduser
```

For more information on adding an Active Directory admin, see [Provision an Azure Active Directory administrator for your server](#)

## Set up Visual Studio

- [Windows client](#)

- [macOS client](#)

1. Visual Studio for Windows is integrated with Azure AD authentication. To enable development and debugging in Visual Studio, add your Azure AD user in Visual Studio by selecting **File > Account Settings** from the menu, and click **Add an account**.
2. To set the Azure AD user for Azure service authentication, select **Tools > Options** from the menu, then select **Azure Service Authentication > Account Selection**. Select the Azure AD user you added and click **OK**.

You're now ready to develop and debug your app with the SQL Database as the back end, using Azure AD authentication.

## Modify your project

The steps you follow for your project depends on whether it's an ASP.NET project or an ASP.NET Core project.

- [ASP.NET](#)
- [ASP.NET Core](#)

1. In Visual Studio, open the Package Manager Console and add the NuGet package [Microsoft.Azure.Services.AppAuthentication](#):

```
Install-Package Microsoft.Azure.Services.AppAuthentication -Version 1.4.0
```

2. In *Web.config*, working from the top of the file and make the following changes:

- In `<configSections>`, add the following section declaration in it:

```
<section name="SqlAuthenticationProviders"
type="System.Data.SqlClient.SqlAuthenticationProviderConfigurationSection, System.Data,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
```

- below the closing `</configSections>` tag, add the following XML code for `<SqlAuthenticationProviders>`.

```
<SqlAuthenticationProviders>
<providers>
<add name="Active Directory Interactive"
type="Microsoft.Azure.Services.AppAuthentication.SqlAppAuthenticationProvider,
Microsoft.Azure.Services.AppAuthentication" />
</providers>
</SqlAuthenticationProviders>
```

- Find the connection string called `MyDbConnection` and replace its `ConnectionString` value with  
`"server=tcp:<server-name>.database.windows.net;database=<db-name>;UID=AnyString;Authentication=Active Directory Interactive"`

. Replace `<server-name>` and `<db-name>` with your server name and database name.

### NOTE

The SqlAuthenticationProvider you just registered is based on top of the AppAuthentication library you installed earlier. By default, it uses a system-assigned identity. To leverage a user-assigned identity, you will need to provide an additional configuration. Please see [connection string support](#) for the AppAuthentication library.

That's every thing you need to connect to SQL Database. When debugging in Visual Studio, your code uses the Azure AD user you configured in [Set up Visual Studio](#). You'll set up SQL Database later to allow connection from the managed identity of your App Service app.

3. Type `Ctrl+F5` to run the app again. The same CRUD app in your browser is now connecting to the Azure SQL Database directly, using Azure AD authentication. This setup lets you run database migrations from Visual Studio.

## Use managed identity connectivity

Next, you configure your App Service app to connect to SQL Database with a system-assigned managed identity.

### NOTE

While the instructions in this section are for a system-assigned identity, a user-assigned identity can just as easily be used. To do this, you would need the change the `az webapp identity assign command` to assign the desired user-assigned identity. Then, when creating the SQL user, make sure to use the name of the user-assigned identity resource rather than the site name.

### Enable managed identity on app

To enable a managed identity for your Azure app, use the `az webapp identity assign` command in the Cloud Shell. In the following command, replace `<app-name>`.

```
az webapp identity assign --resource-group myResourceGroup --name <app-name>
```

### NOTE

To enable managed identity for a [deployment slot](#), add `--slot <slot-name>` and use the name of the slot in `<slot-name>`.

Here's an example of the output:

```
{
 "additionalProperties": {},
 "principalId": "21dfa71c-9e6f-4d17-9e90-1d28801c9735",
 "tenantId": "72f988bf-86f1-41af-91ab-2d7cd011db47",
 "type": "SystemAssigned"
}
```

### Grant permissions to managed identity

#### NOTE

If you want, you can add the identity to an [Azure AD group](#), then grant SQL Database access to the Azure AD group instead of the identity. For example, the following commands add the managed identity from the previous step to a new group called *myAzureSQLDBAccessGroup*.

```
groupid=$(az ad group create --display-name myAzureSQLDBAccessGroup --mail-nickname myAzureSQLDBAccessGroup --query objectId --output tsv)
msiobjectid=$(az webapp identity show --resource-group myResourceGroup --name <app-name> --query principalId --output tsv)
az ad group member add --group $groupid --member-id $msiobjectid
az ad group member list -g $groupid
```

1. In the Cloud Shell, sign in to SQL Database by using the SQLCMD command. Replace *<server-name>* with your server name, *<db-name>* with the database name your app uses, and *<aad-user-name>* and *<aad-password>* with your Azure AD user's credentials.

```
sqlcmd -S <server-name>.database.windows.net -d <db-name> -U <aad-user-name> -P "<aad-password>" -G -l 30
```

2. In the SQL prompt for the database you want, run the following commands to grant the permissions your app needs. For example,

```
CREATE USER [<identity-name>] FROM EXTERNAL PROVIDER;
ALTER ROLE db_datareader ADD MEMBER [<identity-name>];
ALTER ROLE db_datawriter ADD MEMBER [<identity-name>];
ALTER ROLE db_ddladmin ADD MEMBER [<identity-name>];
GO
```

*<identity-name>* is the name of the managed identity in Azure AD. If the identity is system-assigned, the name is always the same as the name of your App Service app. For a [deployment slot](#), the name of its system-assigned identity is *<app-name>/slots/<slot-name>*. To grant permissions for an Azure AD group, use the group's display name instead (for example, *myAzureSQLDBAccessGroup*).

3. Type `EXIT` to return to the Cloud Shell prompt.

#### NOTE

The back-end services of managed identities also [maintains a token cache](#) that updates the token for a target resource only when it expires. If you make a mistake configuring your SQL Database permissions and try to modify the permissions *after* trying to get a token with your app, you don't actually get a new token with the updated permissions until the cached token expires.

#### NOTE

Azure Active Directory and managed identities are not supported for on-premises SQL Server.

## Modify connection string

Remember that the same changes you made in *Web.config* or *appsettings.json* works with the managed identity, so the only thing to do is to remove the existing connection string in App Service, which Visual Studio created deploying your app the first time. Use the following command, but replace *<app-name>* with the name of your app.

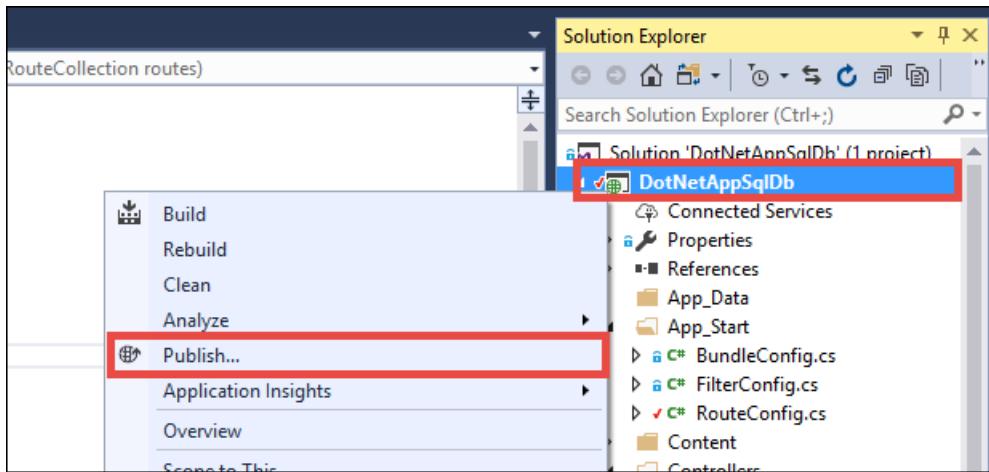
```
az webapp config connection-string delete --resource-group myResourceGroup --name <app-name> --setting-names MyDbConnection
```

## Publish your changes

All that's left now is to publish your changes to Azure.

- [ASP.NET](#)
- [ASP.NET Core](#)

1. If you came from [Tutorial: Build an ASP.NET app in Azure with SQL Database](#), publish your changes in Visual Studio. In the Solution Explorer, right-click your DotNetAppSqlDb project and select Publish.



2. In the publish page, click Publish.

### IMPORTANT

Ensure that your app service name doesn't match with any existing [App Registrations](#). This will lead to Principal ID conflicts.

When the new webpage shows your to-do list, your app is connecting to the database using the managed identity.

Description	Created Date	Done	
Deploy app to Azure	2017-06-01	<input checked="" type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Walk dog	2017-06-03	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Feed cat	2017-06-04	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

© 2017 - My ASP.NET Application

You should now be able to edit the to-do list as before.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

What you learned:

- Enable managed identities
- Grant SQL Database access to the managed identity
- Configure Entity Framework to use Azure AD authentication with SQL Database
- Connect to SQL Database from Visual Studio using Azure AD authentication

[Map an existing custom DNS name to Azure App Service](#)

[Tutorial: Connect to Azure services that don't support managed identities \(using Key Vault\)](#)

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

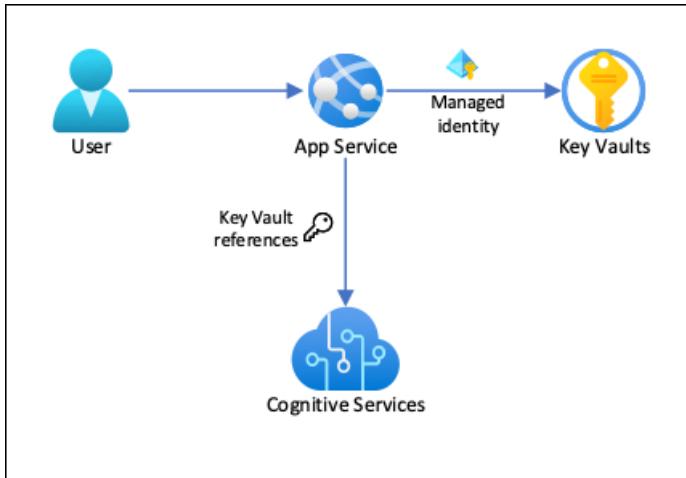
# Tutorial: Secure Cognitive Service connection from App Service using Key Vault

11/2/2021 • 5 minutes to read • [Edit Online](#)

Azure App Service can use managed identities to connect to back-end services without a connection string, which eliminates connection secrets to manage and keeps your back-end connectivity secure in a production environment. For back-end services that don't support managed identities and still require connection secrets, you can use Key Vault to manage connection secrets. This tutorial uses Cognitive Services as an example to show you how it's done in practice. When you're finished, you have an app that makes programmatic calls to Cognitive Services, without storing any connection secrets inside App Service.

## TIP

Azure Cognitive Services do [support authentication through managed identities](#), but this tutorial uses the [subscription key authentication](#) to demonstrate how you could connect to an Azure service that doesn't support managed identities from App Services.



With this architecture:

- Connectivity to Key Vault is secured by managed identities
- App Service accesses the secrets using [Key Vault references](#) as app settings.
- Access to the key vault is restricted to the app. App contributors, such as administrators, may have complete control of the App Service resources, and at the same time have no access to the Key Vault secrets.
- If your application code already accesses connection secrets with app settings, no change is required.

What you will learn:

- Enable managed identities
- Use managed identities to connect to Key Vault
- Use Key Vault references
- Access Cognitive Services

## Prerequisites

Prepare your environment for the Azure CLI.

- Use the Bash environment in [Azure Cloud Shell](#).

[Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

## Create app with connectivity to Cognitive Services

1. Create a resource group to contain all of your resources:

```
Save resource group name as variable for convenience
groupName=myKVResourceGroup
region=westeurope

az group create --name $groupName --location $region
```

2. Create a Cognitive Services resource. Replace <cs-resource-name> with a unique name of your choice.

```
Save resource name as variable for convenience.
cs resourceName=<cs-resource-name>

az cognitiveservices account create --resource-group $groupName --name $cs resourceName --location
$region --kind TextAnalytics --sku F0 --custom-domain $cs resourceName
```

### NOTE

`--sku F0` creates a free tier Cognitive Services resource. Each subscription is limited to a quota of one free-tier `TextAnalytics` resource. If you're already over the quota, use `--sku S` instead.

3. Clone the sample repository locally and deploy the sample application to App Service. Replace <app-name> with a unique name.

- [.NET 5](#)
- [PHP](#)

```
Save app name as variable for convenience
appName=<app-name>

Clone sample application
git clone https://github.com/Azure-Samples/app-service-language-detector.git
cd app-service-language-detector/dotnet

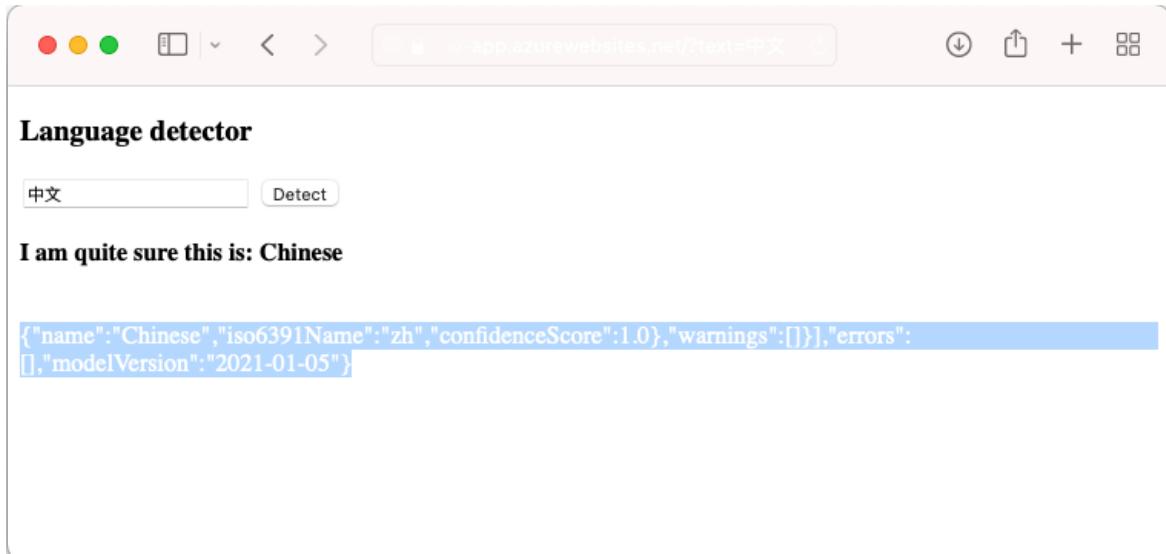
az webapp up --sku F1 --resource-group $groupName --name $appName --plan $appName --location $region
```

4. Configure the Cognitive Services secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY`.

```
Get subscription key for Cognitive Services resource
csKey1=$(az cognitiveservices account keys list --resource-group $groupName --name $csResourceName --
query key1 --output tsv)

az webapp config appsettings set --resource-group $groupName --name $appName --settings
CS_ACCOUNT_NAME="$csResourceName" CS_ACCOUNT_KEY="$csKey1"
```

- In the browser, navigate to your deploy app at <app-name>.azurewebsites.net and try out the language detector with strings in various languages.



If you look at the application code, you may notice the debug output for the detection results in the same font color as the background. You can see it by trying to highlight the white space directly below the result.

## Secure back-end connectivity

At the moment, connection secrets are stored as app settings in your App Service app. This approach is already securing connection secrets from your application codebase. However, any contributor who can manage your app can also see the app settings. In this step, you move the connection secrets to a key vault, and lock down access so that only you can manage it and only the App Service app can read it using its managed identity.

- Create a key vault. Replace <vault-name> with a unique name.

```
Save app name as variable for convenience
vaultName=<vault-name>

az keyvault create --resource-group $groupName --name $vaultName --location $region --sku standard --
enable-rbac-authorization
```

The `--enable-rbac-authorization` parameter [sets Azure role-based access control \(RBAC\) as the permission model](#). This setting by default invalidates all access policies permissions.

- Give yourself the *Key Vault Secrets Officer* RBAC role for the vault.

```
vaultResourceId=$(az keyvault show --name $vaultName --query id --output tsv)
myId=$(az ad signed-in-user show --query objectId --output tsv)
az role assignment create --role "Key Vault Secrets Officer" --assignee-object-id $myId --assignee-
principal-type User --scope $vaultResourceId
```

- Enable the system-assigned managed identity for your app, and give it the *Key Vault Secrets User* RBAC

role for the vault.

```
az webapp identity assign --resource-group $groupName --name $appName --scope $vaultResourceId --role "Key Vault Secrets User"
```

4. Add the Cognitive Services resource name and subscription key as secrets to the vault, and save their IDs as environment variables for the next step.

```
csResourceKVUri=$(az keyvault secret set --vault-name $vaultName --name csresource --value $csResourceName --query id --output tsv)
csKeyKVUri=$(az keyvault secret set --vault-name $vaultName --name cskey --value $csKey1 --query id -o output tsv)
```

5. Previously, you set the secrets as app settings `CS_ACCOUNT_NAME` and `CS_ACCOUNT_KEY` in your app. Now, set them as [key vault references](#) instead.

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)"
CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

6. In the browser, navigate to `<app-name>.azurewebsites.net` again. If you get detection results back, then you're connecting to the Cognitive Services endpoint with key vault references.

Congratulations, your app is now connecting to Cognitive Services using secrets kept in your key vault, without any changes to your application code.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name $groupName
```

This command may take a minute to run.

## Next steps

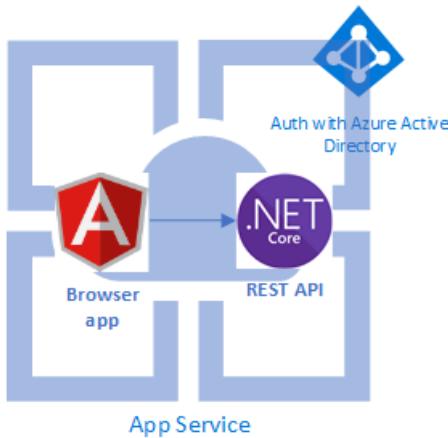
- [Tutorial: Isolate back-end communication with Virtual Network integration](#)
- [Integrate your app with an Azure virtual network](#)
- [App Service networking features](#)

# Tutorial: Authenticate and authorize users end-to-end in Azure App Service

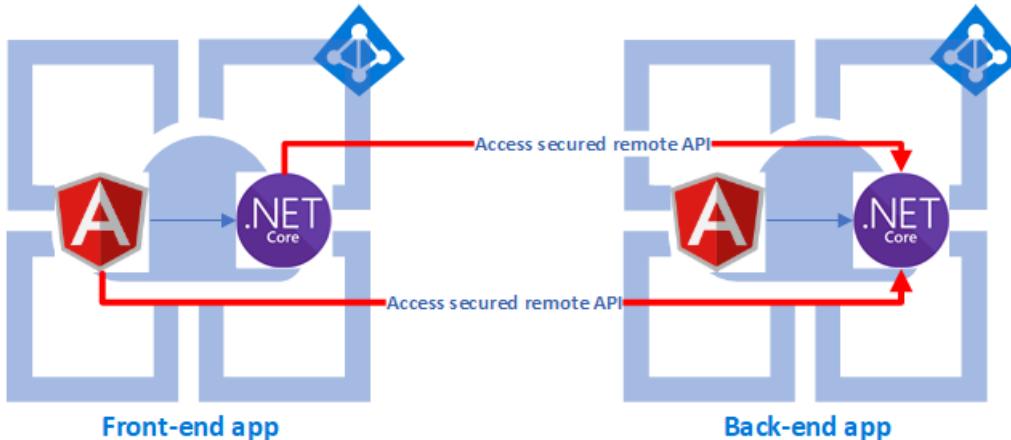
11/2/2021 • 16 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. In addition, App Service has built-in support for [user authentication and authorization](#). This tutorial shows how to secure your apps with App Service authentication and authorization. It uses a ASP.NET Core app with an Angularjs front end as an example. App Service authentication and authorization support all language runtimes, and you can learn how to apply it to your preferred language by following the tutorial.

Azure App Service provides a highly scalable, self-patching web hosting service using the Linux operating system. In addition, App Service has built-in support for [user authentication and authorization](#). This tutorial shows how to secure your apps with App Service authentication and authorization. It uses an ASP.NET Core app with an Angularjs front end as an example. App Service authentication and authorization support all language runtimes, and you can learn how to apply it to your preferred language by following the tutorial.



It also shows you how to secure a multi-tiered app, by accessing a secured back-end API on behalf of the authenticated user, both [from server code](#) and [from browser code](#).



These are only some of the possible authentication and authorization scenarios in App Service.

Here's a more comprehensive list of things you learn in the tutorial:

- Enable built-in authentication and authorization
- Secure apps against unauthenticated requests

- Use Azure Active Directory as the identity provider
- Access a remote app on behalf of the signed-in user
- Secure service-to-service calls with token authentication
- Use access tokens from server code
- Use access tokens from client (browser) code

You can follow the steps in this tutorial on macOS, Linux, Windows.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install the latest .NET Core 3.1 SDK](#)
- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Create local .NET Core app

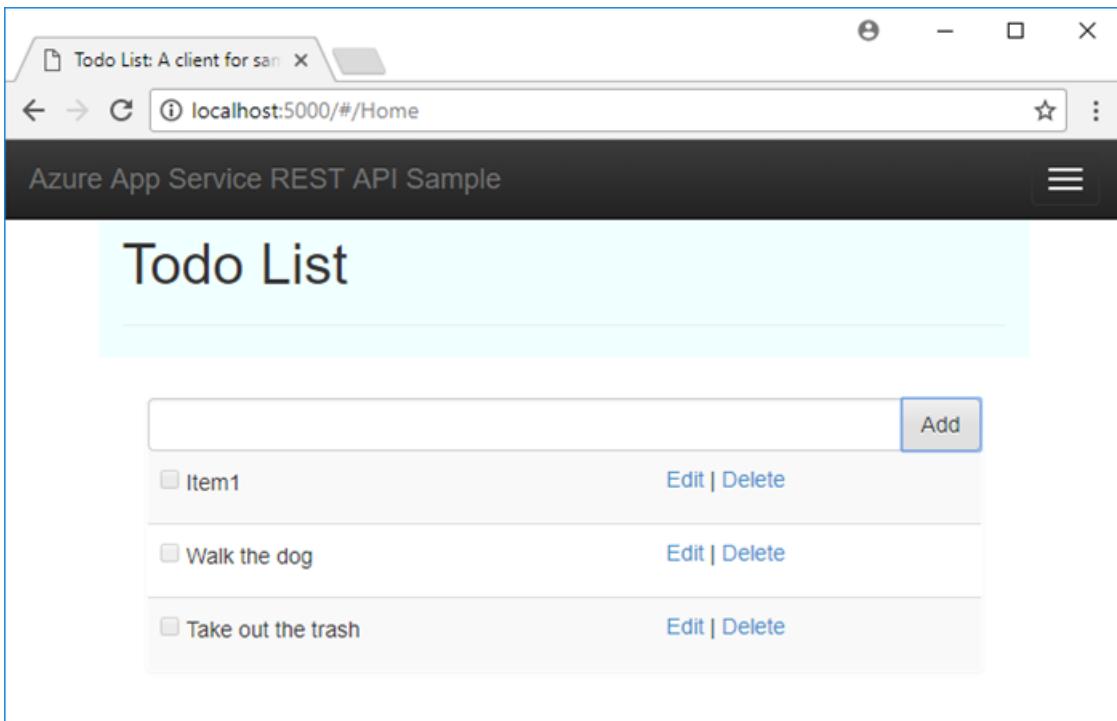
In this step, you set up the local .NET Core project. You use the same project to deploy a back-end API app and a front-end web app.

### Clone and run the sample application

1. Run the following commands to clone the sample repository and run it.

```
git clone https://github.com/Azure-Samples/dotnet-core-api
cd dotnet-core-api
dotnet run
```

2. Navigate to <http://localhost:5000> and try adding, editing, and removing todo items.



3. To stop ASP.NET Core, press `Ctrl+C` in the terminal.

4. Make sure the default branch is `main`.

```
git branch -m main
```

#### TIP

The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main`, this tutorial also shows you how to deploy a repository from `main`. For more information, see [Change deployment branch](#).

## Deploy apps to Azure

In this step, you deploy the project to two App Service apps. One is the front-end app and the other is the back-end app.

### Configure a deployment user

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the `az webapp deployment user set` command in Azure Cloud Shell. Replace `<username>` and `<password>` with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

## Create Azure resources

In the Cloud Shell, run the following commands to create two Windows web apps. Replace <front-end-app-name> and <back-end-app-name> with two globally unique app names (valid characters are a-z, 0-9, and -). For more information on each command, see [Host a RESTful API with CORS in Azure App Service](#).

```
az group create --name myAuthResourceGroup --location "West Europe"
az appservice plan create --name myAuthAppServicePlan --resource-group myAuthResourceGroup --sku FREE
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <front-end-app-name> --deployment-local-git --query deploymentLocalGitUrl
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <back-end-app-name> --deployment-local-git --query deploymentLocalGitUrl
```

In the Cloud Shell, run the following commands to create two web apps. Replace <front-end-app-name> and <back-end-app-name> with two globally unique app names (valid characters are a-z, 0-9, and -). For more information on each command, see [Create a .NET Core app in Azure App Service](#).

```
az group create --name myAuthResourceGroup --location "West Europe"
az appservice plan create --name myAuthAppServicePlan --resource-group myAuthResourceGroup --sku FREE --is-linux
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <front-end-app-name> --runtime "DOTNETCORE|3.1" --deployment-local-git --query deploymentLocalGitUrl
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <back-end-app-name> --runtime "DOTNETCORE|3.1" --deployment-local-git --query deploymentLocalGitUrl
```

### NOTE

Save the URLs of the Git remotes for your front-end app and back-end app, which are shown in the output from

```
az webapp create .
```

## Push to Azure from Git

1. Since you're deploying the main branch, you need to set the default deployment branch for your two App Service apps to main (see [Change deployment branch](#)). In the Cloud Shell, set the DEPLOYMENT\_BRANCH app setting with the az webapp config appsettings set command.

```
az webapp config appsettings set --name <front-end-app-name> --resource-group myAuthResourceGroup --settings DEPLOYMENT_BRANCH=main
az webapp config appsettings set --name <back-end-app-name> --resource-group myAuthResourceGroup --settings DEPLOYMENT_BRANCH=main
```

2. Back in the local terminal window, run the following Git commands to deploy to the back-end app. Replace <deploymentLocalGitUrl-of-back-end-app> with the URL of the Git remote that you saved from [Create Azure resources](#). When prompted for credentials by Git Credential Manager, make sure that you enter [your deployment credentials](#), not the credentials you use to sign in to the Azure portal.

```
git remote add backend <deploymentLocalGitUrl-of-back-end-app>
git push backend main
```

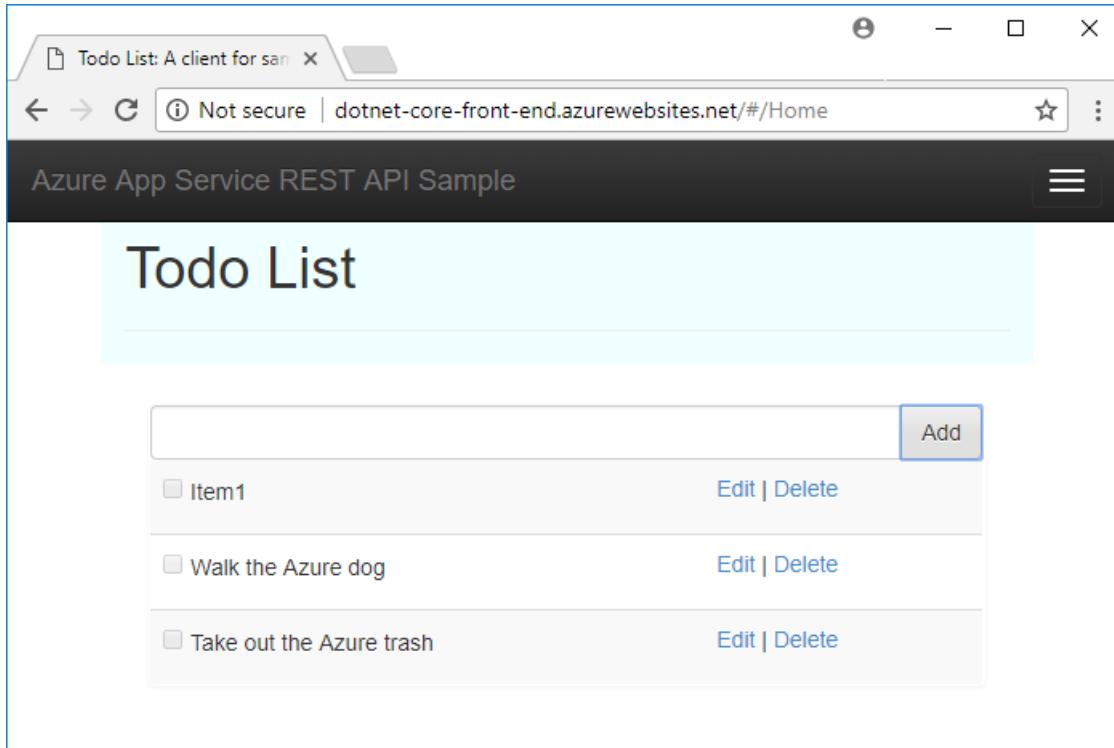
3. In the local terminal window, run the following Git commands to deploy the same code to the front-end app. Replace <deploymentLocalGitUrl-of-front-end-app> with the URL of the Git remote that you saved from [Create Azure resources](#).

```
git remote add frontend <deploymentLocalGitUrl-of-front-end-app>
git push frontend main
```

## Browse to the apps

Navigate to the following URLs in a browser and see the two apps working.

```
http://<back-end-app-name>.azurewebsites.net
http://<front-end-app-name>.azurewebsites.net
```



### NOTE

If your app restarts, you may have noticed that new data has been erased. This behavior by design because the sample ASP.NET Core app uses an in-memory database.

## Call back-end API from front end

In this step, you point the front-end app's server code to access the back-end API. Later, you enable authenticated access from the front end to the back end.

### Modify front-end code

1. In the local repository, open `Controllers/TodoController.cs`. At the beginning of the `TodoController` class, add the following lines and replace `<back-end-app-name>` with the name of your back-end app:

```
private static readonly HttpClient _client = new HttpClient();
private static readonly string _remoteUrl = "https://<back-end-app-name>.azurewebsites.net";
```

2. Find the method that's decorated with `[HttpGet]` and replace the code inside the curly braces with:

```
var data = await _client.GetStringAsync($"{_remoteUrl}/api/Todo");
return JsonConvert.DeserializeObject<List<TodoItem>>(data);
```

The first line makes a `GET /api/Todo` call to the back-end API app.

3. Next, find the method that's decorated with `[HttpGet("{id}")]` and replace the code inside the curly braces with:

```
var data = await _client.GetStringAsync($" {_remoteUrl}/api/Todo/{id}");
return Content(data, "application/json");
```

The first line makes a `GET /api/Todo/{id}` call to the back-end API app.

4. Next, find the method that's decorated with `[HttpPost]` and replace the code inside the curly braces with:

```
var response = await _client.PostAsJsonAsync($" {_remoteUrl}/api/Todo", todoItem);
var data = await response.Content.ReadAsStringAsync();
return Content(data, "application/json");
```

The first line makes a `POST /api/Todo` call to the back-end API app.

5. Next, find the method that's decorated with `[HttpPut("{id}")]` and replace the code inside the curly braces with:

```
var res = await _client.PutAsJsonAsync($" {_remoteUrl}/api/Todo/{id}", todoItem);
return new NoContentResult();
```

The first line makes a `PUT /api/Todo/{id}` call to the back-end API app.

6. Next, find the method that's decorated with `[HttpDelete("{id}")]` and replace the code inside the curly braces with:

```
var res = await _client.DeleteAsync($" {_remoteUrl}/api/Todo/{id}");
return new NoContentResult();
```

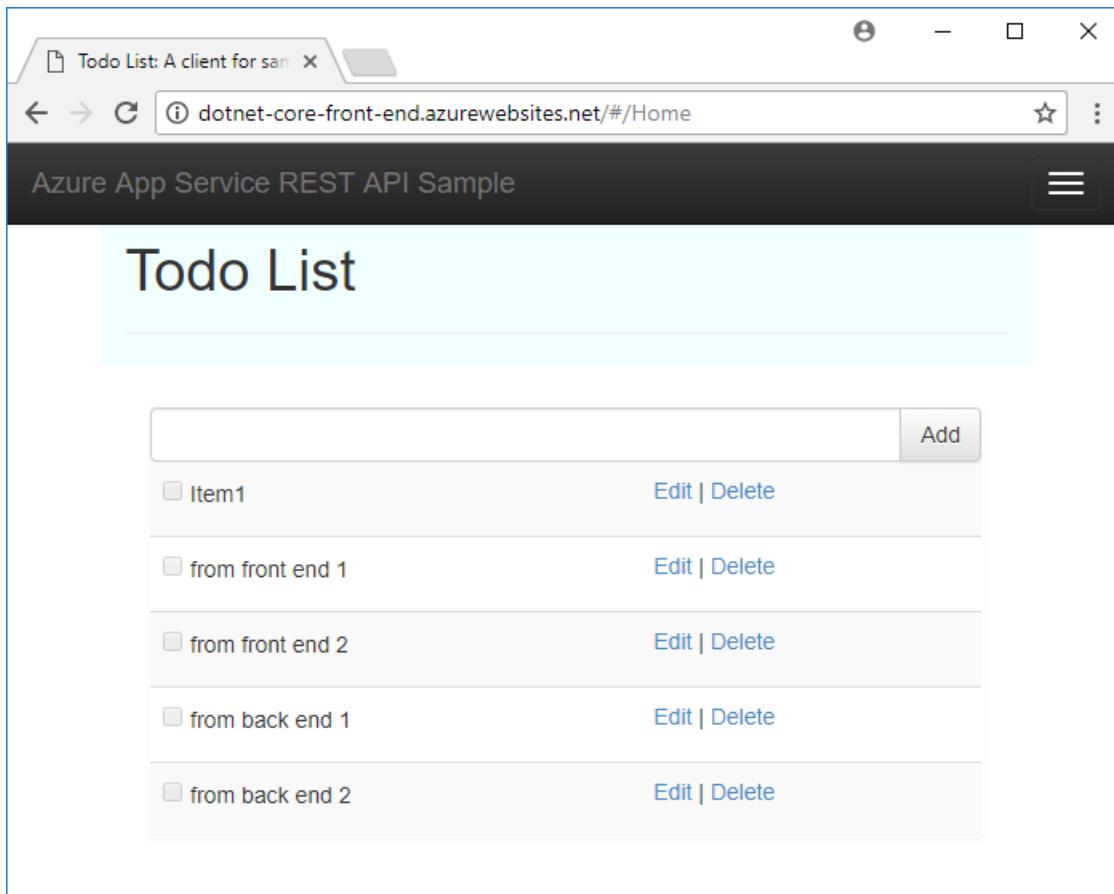
The first line makes a `DELETE /api/Todo/{id}` call to the back-end API app.

7. Save all your changes. In the local terminal window, deploy your changes to the front-end app with the following Git commands:

```
git add .
git commit -m "call back-end API"
git push frontend main
```

## Check your changes

1. Navigate to `http://<front-end-app-name>.azurewebsites.net` and add a few items, such as `from front end 1` and `from front end 2`.
2. Navigate to `http://<back-end-app-name>.azurewebsites.net` to see the items added from the front-end app. Also, add a few items, such as `from back end 1` and `from back end 2`, then refresh the front-end app to see if it reflects the changes.



## Configure auth

In this step, you enable authentication and authorization for the two apps. You also configure the front-end app to generate an access token that you can use to make authenticated calls to the back-end app.

You use Azure Active Directory as the identity provider. For more information, see [Configure Azure Active Directory authentication for your App Services application](#).

### Enable authentication and authorization for back-end app

1. In the [Azure portal](#) menu, select **Resource groups** or search for and select **Resource groups** from any page.
2. In **Resource groups**, find and select your resource group. In **Overview**, select your back-end app's management page.

The screenshot shows the Azure portal's "Resource groups" blade. A resource group named "myAuthResourceGroup" is selected. In the main pane, there is a list of resources under the heading "Essentials". One item, "dotnet-core-back-end-auth", is highlighted with a red box. The list includes:

Name	Type
dotnet-core-back-end-auth	App Service
dotnet-core-front-end-auth	App Service
myAuthAppServicePlan	App Service plan

3. In your back-end app's left menu, select **Authentication**, and then click **Add identity provider**.
4. In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Azure AD identities.

5. Accept the default settings and click **Add**.

Home > dotnet-core-back-end-auth >

## Add an identity provider ...

**Basics** Permissions

Identity provider \* Microsoft

**App registration**

An app registration associates your identity provider with your app. Enter the app registration information here, or go to your provider to create a new one. [Learn more ↗](#)

App registration type \*  Create new app registration  Pick an existing app registration in this directory  Provide the details of an existing app registration

Name \* dotnet-core-back-end-auth

Supported account types \*  Current tenant - Single tenant  Any Azure AD directory - Multi-tenant  Any Azure AD directory & personal Microsoft accounts  Personal Microsoft accounts only [Help me choose...](#)

**App Service authentication settings**

Requiring authentication ensures all users of your app will need to authenticate. If you allow unauthenticated requests, you'll need your own code for specific authentication requirements. [Learn more ↗](#)

Authentication \*  Require authentication  Allow unauthenticated access

Unauthenticated requests \*  HTTP 302 Found redirect: recommended for websites  HTTP 401 Unauthorized: recommended for APIs  HTTP 403 Forbidden

Redirect to \* Microsoft

Token store

**Add** < Previous Next: Permissions >

6. The **Authentication** page opens. Copy the **Client ID** of the Azure AD application to a notepad. You need this value later.

With App Service you can choose an identity provider to manage user identities and authentication flows. Add providers here, edit settings, and decide which provider is handling authentication for your app. [Learn more](#)

Identity provider	App (client) ID	Learn more	Edit	Delete
Microsoft ( <a href="#">dotnet-core-back-end-auth</a> )	462263c1-2c67-4538-ab6a-1514442a82ed	<a href="#">Quickstart</a>	<a href="#">Edit</a>	<a href="#">Delete</a>

If you stop here, you have a self-contained app that's already secured by the App Service authentication and authorization. The remaining sections show you how to secure a multi-app solution by "flowing" the authenticated user from the front end to the back end.

### Enable authentication and authorization for front-end app

Follow the same steps for the front-end app, but skip the last step. You don't need the client ID for the front-end app. However, stay on the **Authentication** page for the front-end app because you'll use it in the next step.

If you like, navigate to `http://<front-end-app-name>.azurewebsites.net`. It should now direct you to a secured sign-in page. After you sign in, *you still can't access the data from the back-end app*, because the back-end app now requires Azure Active Directory sign-in from the front-end app. You need to do three things:

- Grant the front end access to the back end
- Configure App Service to return a usable token
- Use the token in your code

#### TIP

If you run into errors and reconfigure your app's authentication/authorization settings, the tokens in the token store may not be regenerated from the new settings. To make sure your tokens are regenerated, you need to sign out and sign back in to your app. An easy way to do it is to use your browser in private mode, and close and reopen the browser in private mode after changing the settings in your apps.

### Grant front-end app access to back end

Now that you've enabled authentication and authorization to both of your apps, each of them is backed by an AD application. In this step, you give the front-end app permissions to access the back end on the user's behalf. (Technically, you give the front end's *AD application* the permissions to access the back end's *AD application* on the user's behalf.)

1. In the **Authentication** page for the front-end app, select your front-end app name under **Identity provider**. This app registration was automatically generated for you. Select **API permissions** in the left menu.
2. Select **Add a permission**, then select **My APIs > <back-end-app-name>**.
3. In the **Request API permissions** page for the back-end app, select **Delegated permissions** and **user\_impersonation**, then select **Add permissions**.

## Request API permissions

[All APIs](#)

 dotnet-core-back-end-auth  
api://XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX

What type of permissions does your application require?

Delegated permissions  
Your application needs to access the API as the signed-in user.

Application permissions  
Your application runs as a background service or daemon without a signed-in user.

Select permissions expand all

Start typing a permission to filter these results

i The "Admin consent required" column shows the default value for an organization. However, user consent can be customized per permission, user, or app. This column may not reflect the value in your organization, or in organizations where this app will be used. [Learn more](#)

Permission	Admin consent required
Permissions (1)	
<input checked="" type="checkbox"/> user_impersonation <input type="radio"/>	No
Access dotnet-core-back-end-auth	

Add permissions Discard

## Configure App Service to return a usable access token

The front-end app now has the required permissions to access the back-end app as the signed-in user. In this step, you configure App Service authentication and authorization to give you a usable access token for accessing the back end. For this step, you need the back end's client ID, which you copied from [Enable authentication and authorization for back-end app](#).

In the Cloud Shell, run the following command on the front-end app to add the `scope` parameter to the authentication setting `identityProviders.azureActiveDirectory.login.loginParameters`. Replace `<front-end-app-name>` and `<back-end-client-id>`.

```
az webapp auth set --resource-group myAuthResourceGroup --name <front-end-app-name> --body
'{"identityProviders":{"azureActiveDirectory":{"login":{"loginParameters":["scope=openid profile email
offline_access api://<back-end-client-id>/user_impersonation"]}}}'
```

Here's an explanation of the requested scopes:

- `openid`, `profile`, and `email` are requested by App Service by default already. For information, see [OpenID Connect Scopes](#).
- `api://<back-end-client-id>/user_impersonation` is an exposed API in your back-end app registration. It's the scope that gives you a JWT token that includes the back end app as a `token audience`.
- `offline_access` is included here for convenience (in case you want to [refresh tokens](#)).

## TIP

- To view the `api://<back-end-client-id>/user_impersonation` scope in the Azure portal, go to the **Authentication** page for the back-end app, click the link under **Identity provider**, then click **Expose an API** in the left menu.
- To configure the required scopes using a web interface instead, see the Microsoft steps at [Refresh auth tokens](#).
- Some scopes require admin or user consent. This requirement causes the consent request page to be displayed when a user signs into the front-end app in the browser. To avoid this consent page, add the front end's app registration as an authorized client application in the **Expose an API** page by clicking **Add a client application** and supplying the client ID of the front end's app registration.

## NOTE

For Linux apps, There's a temporary requirement to configure a versioning setting for the back-end app registration. In the Cloud Shell, configure it with the following commands. Be sure to replace `<back-end-client-id>` with your back end's client ID.

```
id=$(az ad app show --id <back-end-client-id> --query objectId --output tsv)
az rest --method PATCH --url https://graph.microsoft.com/v1.0/applications/$id --body "{'api':
{'requestedAccessTokenVersion':2}}"
```

Your apps are now configured. The front end is now ready to access the back end with a proper access token.

For information on how to configure the access token for other providers, see [Refresh identity provider tokens](#).

## Call API securely from server code

In this step, you enable your previously modified server code to make authenticated calls to the back-end API.

Your front-end app now has the required permission and also adds the back end's client ID to the login parameters. Therefore, it can obtain an access token for authentication with the back-end app. App Service supplies this token to your server code by injecting a `X-MS-TOKEN-AAD-ACCESS-TOKEN` header to each authenticated request (see [Retrieve tokens in app code](#)).

## NOTE

These headers are injected for all supported languages. You access them using the standard pattern for each respective language.

1. In the local repository, open `Controllers/TodoController.cs` again. Under the `TodoController(TodoContext context)` constructor, add the following code:

```
public override void OnActionExecuting(ActionExecutingContext context)
{
 base.OnActionExecuting(context);

 _client.DefaultRequestHeaders.Accept.Clear();
 _client.DefaultRequestHeaders.Authorization =
 new AuthenticationHeaderValue("Bearer", Request.Headers["X-MS-TOKEN-AAD-ACCESS-TOKEN"]);
}
```

This code adds the standard HTTP header `Authorization: Bearer <access-token>` to all remote API calls. In the ASP.NET Core MVC request execution pipeline, `OnActionExecuting` executes just before the respective

action does, so each of your outgoing API call now presents the access token.

2. Save all your changes. In the local terminal window, deploy your changes to the front-end app with the following Git commands:

```
git add .
git commit -m "add authorization header for server code"
git push frontend main
```

3. Sign in to <https://<front-end-app-name>.azurewebsites.net> again. At the user data usage agreement page, click **Accept**.

You should now be able to create, read, update, and delete data from the back-end app as before. The only difference now is that both apps are now secured by App Service authentication and authorization, including the service-to-service calls.

Congratulations! Your server code is now accessing the back-end data on behalf of the authenticated user.

## Call API securely from browser code

In this step, you point the front-end Angularjs app to the back-end API. This way, you learn how to retrieve the access token and make API calls to the back-end app with it.

While the server code has access to request headers, client code can access `GET /.auth/me` to get the same access tokens (see [Retrieve tokens in app code](#)).

### TIP

This section uses the standard HTTP methods to demonstrate the secure HTTP calls. However, you can use [Microsoft Authentication Library for JavaScript](#) to help simplify the Angularjs application pattern.

### Configure CORS

In the Cloud Shell, enable CORS to your client's URL by using the `az webapp cors add` command. Replace the `<back-end-app-name>` and `<front-end-app-name>` placeholders.

```
az webapp cors add --resource-group myAuthResourceGroup --name <back-end-app-name> --allowed-origins
'https://<front-end-app-name>.azurewebsites.net'
```

This step is not related to authentication and authorization. However, you need it so that your browser allows the cross-domain API calls from your Angularjs app. For more information, see [Add CORS functionality](#).

### Point Angular.js app to back-end API

1. In the local repository, open `wwwroot/index.html`.
2. In Line 51, set the `apiEndpoint` variable to the HTTPS URL of your back-end app (<https://<back-end-app-name>.azurewebsites.net>). Replace `<back-end-app-name>` with your app name in App Service.
3. In the local repository, open `wwwroot/app/scripts/todoListSvc.js` and see that `apiEndpoint` is prepended to all the API calls. Your Angularjs app is now calling the back-end APIs.

### Add access token to API calls

1. In `wwwroot/app/scripts/todoListSvc.js`, above the list of API calls (above the line `getItems : function(){},`) add the following function to the list:

```
setAuth: function (token) {
 $http.defaults.headers.common['Authorization'] = 'Bearer ' + token;
},
```

This function is called to set the default `Authorization` header with the access token. You call it in the next step.

2. In the local repository, open `wwwroot/app/scripts/app.js` and find the following code:

```
$routeProvider.when("/Home", {
 controller: "todoListCtrl",
 templateUrl: "/App/Views/TodoList.html",
}).otherwise({ redirectTo: "/Home" });
```

3. Replace the entire code block with the following code:

```
$routeProvider.when("/Home", {
 controller: "todoListCtrl",
 templateUrl: "/App/Views/TodoList.html",
 resolve: {
 token: ['$http', 'todoListSvc', function ($http, todoListSvc) {
 return $http.get('/.auth/me').then(function (response) {
 todoListSvc.setAuth(response.data[0].access_token);
 return response.data[0].access_token;
 });
 }]
 },
}).otherwise({ redirectTo: "/Home" });
```

The new change adds the `resolve` mapping that calls `/auth/me` and sets the access token. It makes sure you have the access token before instantiating the `todoListCtrl` controller. That way all API calls by the controller includes the token.

## Deploy updates and test

1. Save all your changes. In the local terminal window, deploy your changes to the front-end app with the following Git commands:

```
git add .
git commit -m "add authorization header for Angular"
git push frontend main
```

2. Navigate to `https://<front-end-app-name>.azurewebsites.net` again. You should now be able to create, read, update, and delete data from the back-end app, directly in the Angular.js app.

Congratulations! Your client code is now accessing the back-end data on behalf of the authenticated user.

## When access tokens expire

Your access token expires after some time. For information on how to refresh your access tokens without requiring users to reauthenticate with your app, see [Refresh identity provider tokens](#).

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myAuthResourceGroup
```

This command may take a minute to run.

## Next steps

What you learned:

- Enable built-in authentication and authorization
- Secure apps against unauthenticated requests
- Use Azure Active Directory as the identity provider
- Access a remote app on behalf of the signed-in user
- Secure service-to-service calls with token authentication
- Use access tokens from server code
- Use access tokens from client (browser) code

Advance to the next tutorial to learn how to map a custom DNS name to your app.

[Map an existing custom DNS name to Azure App Service](#)

# Tutorial: Enable authentication in App Service and access storage and Microsoft Graph

11/2/2021 • 2 minutes to read • [Edit Online](#)

This tutorial describes a common application scenario in which you learn how to:

- [Configure authentication for a web app](#) and limit access to users in your organization. See A in the diagram.
- [Securely access Azure Storage](#) for the web application using managed identities. See B in the diagram.
- Access data in Microsoft Graph [for the signed-in user](#) or [for the web application](#) using managed identities. See C in the diagram.
- [Clean up the resources](#) you created for this tutorial.

To begin, learn how to enable authentication for a web app.

[Configure authentication for a web app](#)

# Tutorial: Add authentication to your web app running on Azure App Service

11/2/2021 • 3 minutes to read • [Edit Online](#)

Learn how to enable authentication for your web app running on Azure App Service and limit access to users in your organization.

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. Using the App Service authentication/authorization module isn't required, but helps simplify authentication and authorization for your app. This article shows how to secure your web app with the App Service authentication/authorization module by using Azure Active Directory (Azure AD) as the identity provider.

The authentication/authorization module is enabled and configured through the Azure portal and app settings. No SDKs, specific languages, or changes to application code are required. A variety of identity providers are supported, which includes Azure AD, Microsoft Account, Facebook, Google, and Twitter. When the authentication/authorization module is enabled, every incoming HTTP request passes through it before being handled by app code. To learn more, see [Authentication and authorization in Azure App Service](#).

In this tutorial, you learn how to:

- Configure authentication for the web app.
- Limit access to the web app to users in your organization.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create and publish a web app on App Service

For this tutorial, you need a web app deployed to App Service. You can use an existing web app, or you can follow the [ASP.NET Core quickstart](#) to create and publish a new web app to App Service.

Whether you use an existing web app or create a new one, take note of the web app name and the name of the resource group that the web app is deployed to. You need these names throughout this tutorial.

## Configure authentication and authorization

You now have a web app running on App Service. Next, you enable authentication and authorization for the web app. You use Azure AD as the identity provider. For more information, see [Configure Azure AD authentication for your App Service application](#).

In the [Azure portal](#) menu, select **Resource groups**, or search for and select **Resource groups** from any page.

In **Resource groups**, find and select your resource group. In **Overview**, select your app's management page.

The screenshot shows the Azure Resource Groups blade. On the left, there's a sidebar with navigation links like Home, Resource groups, Fourth Coffee, Create, Manage view, Filter for any field..., and a search bar. The main area is titled 'WebApp-EasyAuth-DotNet20210401164235ResourceGroup' and shows an 'Overview' section with tabs for Activity log, Access control (IAM), Tags, Events, Deployments, Security, Policies, Properties, Locks, Cost Management, and Cost analysis. Below the tabs, there's a table listing resources:

Name	Type	Location
securewebappstorage	Storage account	South Central US
WebApp-EasyAuth-DotNet20210401164235	App Service	South Central US
WebApp-EasyAuth-DotNet20210401164235Plan	App Service plan	South Central US

On your app's left menu, select **Authentication**, and then click **Add identity provider**.

In the **Add an identity provider** page, select **Microsoft** as the **Identity provider** to sign in Microsoft and Azure AD identities.

For **App registration > App registration type**, select **Create new app registration**.

For **App registration > Supported account types**, select **Current tenant-single tenant**.

In the **App Service authentication settings** section, leave **Authentication** set to **Require authentication** and **Unauthenticated requests** set to **HTTP 302 Found redirect: recommended for websites**.

At the bottom of the **Add an identity provider** page, click **Add** to enable authentication for your web app.

## Add an identity provider ...

[Basics](#) [Permissions](#)

Identity provider \*

Microsoft



### App registration

An app registration associates your identity provider with your app. Enter the app registration information here, or go to your provider to create a new one. [Learn more ↗](#)

App registration type \*

Create new app registration

Pick an existing app registration in this directory

Provide the details of an existing app registration

Name \*

WebApp-EasyAuth-DotNet20210401164235



Supported account types \*

Current tenant - Single tenant

Any Azure AD directory - Multi-tenant

Any Azure AD directory & personal Microsoft accounts

Personal Microsoft accounts only

[Help me choose...](#)

### App Service authentication settings

Requiring authentication ensures all users of your app will need to authenticate. If you allow unauthenticated requests, you'll need your own code for specific authentication requirements. [Learn more ↗](#)

Authentication \*

Require authentication

Allow unauthenticated access

Unauthenticated requests \*

HTTP 302 Found redirect: recommended for websites

HTTP 401 Unauthorized: recommended for APIs

HTTP 403 Forbidden

Redirect to

Microsoft



Token store ⓘ



[Add](#)

< Previous

Next: Permissions >

You now have an app that's secured by the App Service authentication and authorization.

#### NOTE

To allow accounts from other tenants, change the 'Issuer URL' to '<https://login.microsoftonline.com/common/v2.0>' by editing your 'Identity Provider' from the 'Authentication' blade.

## Verify limited access to the web app

When you enabled the App Service authentication/authorization module, an app registration was created in

your Azure AD tenant. The app registration has the same display name as your web app. To check the settings, select **Azure Active Directory** from the portal menu, and select **App registrations**. Select the app registration that was created. In the overview, verify that **Supported account types** is set to **My organization only**.

Home > Fourth Coffee >

The screenshot shows the Azure portal's 'Overview' page for an app registration. The app's display name is 'WebApp-EasyAuth-DotNet20210401164235'. On the left, there's a sidebar with 'Overview', 'Quickstart', 'Integration assistant', 'Manage' (with 'Branding', 'Authentication', 'Certificates & secrets', and 'Token configuration' options), and a feedback message. The main area shows 'Essentials' details: Display name, Application (client) ID, Directory (tenant) ID, and Object ID. On the right, there's a 'Supported account types' section with a red box around 'My organization only'. Below it are 'Redirect URLs' (1 web, 0 spa, 0 public client), 'Application ID URI' (with a 'Add an Application ID URI' link), and 'Managed application in local directory' (with a link to 'WebApp-EasyAuth-DotNet20210401164235').

To verify that access to your app is limited to users in your organization, start a browser in incognito or private mode and go to `https://<app-name>.azurewebsites.net`. You should be directed to a secured sign-in page, verifying that unauthenticated users aren't allowed access to the site. Sign in as a user in your organization to gain access to the site. You can also start up a new browser and try to sign in by using a personal account to verify that users outside the organization don't have access.

## Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, [clean up the resources you created](#).

## Next steps

In this tutorial, you learned how to:

- Configure authentication for the web app.
- Limit access to the web app to users in your organization.

[App service accesses storage](#)

# Tutorial: Access Azure Storage from a web app

11/2/2021 • 9 minutes to read • [Edit Online](#)

Learn how to access Azure Storage for a web app (not a signed-in user) running on Azure App Service by using managed identities.

You want to add access to the Azure data plane (Azure Storage, Azure SQL Database, Azure Key Vault, or other services) from your web app. You could use a shared key, but then you have to worry about operational security of who can create, deploy, and manage the secret. It's also possible that the key could be checked into GitHub, which hackers know how to scan for. A safer way to give your web app access to data is to use [managed identities](#).

A managed identity from Azure Active Directory (Azure AD) allows App Service to access resources through role-based access control (RBAC), without requiring app credentials. After assigning a managed identity to your web app, Azure takes care of the creation and distribution of a certificate. People don't have to worry about managing secrets or app credentials.

In this tutorial, you learn how to:

- Create a system-assigned managed identity on a web app.
- Create a storage account and an Azure Blob Storage container.
- Access storage from a web app by using managed identities.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

## Enable managed identity on an app

If you create and publish your web app through Visual Studio, the managed identity was enabled on your app for you. In your app service, select **Identity** in the left pane, and then select **System assigned**. Verify that the **Status** is set to **On**. If not, select **Save** and then select **Yes** to enable the system-assigned managed identity. When the managed identity is enabled, the status is set to **On** and the object ID is available.

The screenshot shows the Azure portal interface for a web application named 'SecureWebApp20200924113531'. The left sidebar has a 'Deployment' section with 'Quickstart', 'Deployment slots', 'Deployment Center', and 'Deployment Center (Preview)'. The 'Identity' section is highlighted with a red box. The main content area shows 'System assigned' selected for the identity type, with a note explaining that a system-assigned managed identity enables Azure resources to authenticate. It also shows the 'Status' as 'On', an 'Object ID' of '09382857-ed3f-4cbd-b524-4138df10bfa2', and 'Permissions' as 'Azure role assignments'. A note at the bottom states: 'This resource is registered with Azure Active Directory. You can control it'.

This step creates a new object ID, different than the app ID created in the **Authentication/Authorization** pane. Copy the object ID of the system-assigned managed identity. You'll need it later.

## Create a storage account and Blob Storage container

Now you're ready to create a storage account and Blob Storage container.

Every storage account must belong to an Azure resource group. A resource group is a logical container for grouping your Azure services. When you create a storage account, you have the option to either create a new resource group or use an existing resource group. This article shows how to create a new resource group.

A general-purpose v2 storage account provides access to all of the Azure Storage services: blobs, files, queues, tables, and disks. The steps outlined here create a general-purpose v2 storage account, but the steps to create any type of storage account are similar.

Blobs in Azure Storage are organized into containers. Before you can upload a blob later in this tutorial, you must first create a container.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

To create a general-purpose v2 storage account in the Azure portal, follow these steps.

1. On the Azure portal menu, select **All services**. In the list of resources, enter **Storage Accounts**. As you begin typing, the list filters based on your input. Select **Storage Accounts**.
2. In the **Storage Accounts** window that appears, select **Add**.
3. Select the subscription in which to create the storage account.
4. Under the **Resource group** field, select the resource group that contains your web app from the drop-down menu.
5. Next, enter a name for your storage account. The name you choose must be unique across Azure. The

name also must be between 3 and 24 characters in length and can include numbers and lowercase letters only.

6. Select a location for your storage account, or use the default location.

7. Leave these fields set to their default values:

FIELD	VALUE
Deployment model	Resource Manager
Performance	Standard
Account kind	StorageV2 (general-purpose v2)
Replication	Read-access geo-redundant storage (RA-GRS)
Access tier	Hot

8. Select **Review + Create** to review your storage account settings and create the account.

9. Select **Create**.

To create a Blob Storage container in Azure Storage, follow these steps.

1. Go to your new storage account in the Azure portal.
2. In the left menu for the storage account, scroll to the **Blob service** section, and then select **Containers**.
3. Select the **+ Container** button.
4. Type a name for your new container. The container name must be lowercase, must start with a letter or number, and can include only letters, numbers, and the dash (-) character.
5. Set the level of public access to the container. The default level is **Private (no anonymous access)**.
6. Select **OK** to create the container.

## Grant access to the storage account

You need to grant your web app access to the storage account before you can create, read, or delete blobs. In a previous step, you configured the web app running on App Service with a managed identity. Using Azure RBAC, you can give the managed identity access to another resource, just like any security principal. The Storage Blob Data Contributor role gives the web app (represented by the system-assigned managed identity) read, write, and delete access to the blob container and data.

- [Portal](#)
- [PowerShell](#)
- [Azure CLI](#)

In the [Azure portal](#), go into your storage account to grant your web app access. Select **Access control (IAM)** in the left pane, and then select **Role assignments**. You'll see a list of who has access to the storage account. Now you want to add a role assignment to a robot, the app service that needs access to the storage account. Select **Add > Add role assignment** to open the **Add role assignment** page.

Assign the **Storage Blob Data Contributor** role to the **App Service** at subscription scope. For detailed steps, see [Assign Azure roles using the Azure portal](#).

Your web app now has access to your storage account.

## Access Blob Storage (.NET)

The [DefaultAzureCredential](#) class is used to get a token credential for your code to authorize requests to Azure Storage. Create an instance of the [DefaultAzureCredential](#) class, which uses the managed identity to fetch tokens and attach them to the service client. The following code example gets the authenticated token credential and uses it to create a service client object, which uploads a new blob.

To see this code as part of a sample application, see the [sample on GitHub](#).

### Install client library packages

Install the [Blob Storage NuGet package](#) to work with Blob Storage and the [Azure Identity client library for .NET NuGet package](#) to authenticate with Azure AD credentials. Install the client libraries by using the .NET Core command-line interface or the Package Manager Console in Visual Studio.

- [Command line](#)
- [Package Manager](#)

Open a command line, and switch to the directory that contains your project file.

Run the install commands.

```
dotnet add package Azure.Storage.Blobs
dotnet add package Azure.Identity
```

### Example

```

using System;
using Azure.Storage.Blobs;
using Azure.Storage.Blobs.Models;
using System.Collections.Generic;
using System.Threading.Tasks;
using System.Text;
using System.IO;
using Azure.Identity;

// Some code omitted for brevity.

static public async Task UploadBlob(string accountName, string containerName, string blobName, string
blobContents)
{
 // Construct the blob container endpoint from the arguments.
 string containerEndpoint = string.Format("https://'{0}'.blob.core.windows.net/{1}",
 accountName,
 containerName);

 // Get a credential and create a client object for the blob container.
 BlobContainerClient containerClient = new BlobContainerClient(new Uri(containerEndpoint),
 new DefaultAzureCredential());

 try
 {
 // Create the container if it does not exist.
 await containerClient.CreateIfNotExistsAsync();

 // Upload text to a new block blob.
 byte[] byteArray = Encoding.ASCII.GetBytes(blobContents);

 using (MemoryStream stream = new MemoryStream(byteArray))
 {
 await containerClient.UploadBlobAsync(blobName, stream);
 }
 }
 catch (Exception e)
 {
 throw e;
 }
}

```

## Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, [clean up the resources you created](#).

## Next steps

In this tutorial, you learned how to:

- Create a system-assigned managed identity.
- Create a storage account and Blob Storage container.
- Access storage from a web app by using managed identities.

[Tutorial: Isolate back-end communication with Virtual Network integration](#)

[App Service accesses Microsoft Graph on behalf of the user](#)

[Map an existing custom DNS name to Azure App Service](#)

# Tutorial: Access Microsoft Graph from a secured app as the user

11/2/2021 • 6 minutes to read • [Edit Online](#)

Learn how to access Microsoft Graph from a web app running on Azure App Service.

You want to add access to Microsoft Graph from your web app and perform some action as the signed-in user. This section describes how to grant delegated permissions to the web app and get the signed-in user's profile information from Azure Active Directory (Azure AD).

In this tutorial, you learn how to:

- Grant delegated permissions to a web app.
- Call Microsoft Graph from a web app for a signed-in user.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

## Grant front-end access to call Microsoft Graph

Now that you've enabled authentication and authorization on your web app, the web app is registered with the Microsoft identity platform and is backed by an Azure AD application. In this step, you give the web app permissions to access Microsoft Graph for the user. (Technically, you give the web app's Azure AD application the permissions to access the Microsoft Graph Azure AD application for the user.)

In the [Azure portal](#) menu, select **Azure Active Directory** or search for and select **Azure Active Directory** from any page.

Select **App registrations > Owned applications > View all applications in this directory**. Select your web app name, and then select **API permissions**.

Select **Add a permission**, and then select Microsoft APIs and Microsoft Graph.

Select **Delegated permissions**, and then select **User.Read** from the list. Select **Add permissions**.

## Configure App Service to return a usable access token

The web app now has the required permissions to access Microsoft Graph as the signed-in user. In this step, you configure App Service authentication and authorization to give you a usable access token for accessing Microsoft Graph. For this step, you need to add the User.Read scope for the downstream service (Microsoft Graph): <https://graph.microsoft.com/User.Read>.

### IMPORTANT

If you don't configure App Service to return a usable access token, you receive a `CompactToken parsing failed with error code: 80049217` error when you call Microsoft Graph APIs in your code.

- [Azure Resource Explorer](#)
- [Azure CLI](#)

Go to [Azure Resource Explorer](#) and using the resource tree, locate your web app. The resource URL should be similar to

`https://resources.azure.com/subscriptions/subscriptionId/resourceGroups/SecureWebApp/providers/Microsoft.Web/sites/SecureWebApp2020`

The Azure Resource Explorer is now opened with your web app selected in the resource tree. At the top of the page, select **Read/Write** to enable editing of your Azure resources.

In the left browser, drill down to **config > authsettingsV2**.

In the **authsettingsV2** view, select **Edit**. Find the **login** section of **identityProviders -> azureActiveDirectory** and add the following **loginParameters** settings:

```
"loginParameters": ["response_type=code id_token", "scope=openid offline_access profile https://graph.microsoft.com/User.Read"]
```

```
"identityProviders": {
 "azureActiveDirectory": {
 "enabled": true,
 "login": {
 "loginParameters": [
 "response_type=code id_token",
 "scope=openid offline_access profile https://graph.microsoft.com/User.Read"
]
 }
 }
},
```

Save your settings by selecting **PUT**. This setting can take several minutes to take effect. Your web app is now configured to access Microsoft Graph with a proper access token. If you don't, Microsoft Graph returns an error saying that the format of the compact token is incorrect.

## Call Microsoft Graph (.NET)

Your web app now has the required permissions and also adds Microsoft Graph's client ID to the login parameters. Using the [Microsoft.Identity.Web library](#), the web app gets an access token for authentication with Microsoft Graph. In version 1.2.0 and later, the Microsoft.Identity.Web library integrates with and can run alongside the App Service authentication/authorization module. Microsoft.Identity.Web detects that the web app is hosted in App Service and gets the access token from the App Service authentication/authorization module. The access token is then passed along to authenticated requests with the Microsoft Graph API.

To see this code as part of a sample application, see the [sample on GitHub](#).

### NOTE

The Microsoft.Identity.Web library isn't required in your web app for basic authentication/authorization or to authenticate requests with Microsoft Graph. It's possible to [securely call downstream APIs](#) with only the App Service authentication/authorization module enabled.

However, the App Service authentication/authorization is designed for more basic authentication scenarios. For more complex scenarios (handling custom claims, for example), you need the Microsoft.Identity.Web library or [Microsoft Authentication Library](#). There's a little more setup and configuration work in the beginning, but the Microsoft.Identity.Web library can run alongside the App Service authentication/authorization module. Later, when your web app needs to handle more complex scenarios, you can disable the App Service authentication/authorization module and Microsoft.Identity.Web will already be a part of your app.

### Install client library packages

Install the [Microsoft.Identity.Web](#) and [Microsoft.Identity.Web.MicrosoftGraph](#) NuGet packages in your project by using the .NET Core command-line interface or the Package Manager Console in Visual Studio.

- [Command line](#)
- [Package Manager](#)

Open a command line, and switch to the directory that contains your project file.

Run the install commands.

```
dotnet add package Microsoft.Identity.Web.MicrosoftGraph
dotnet add package Microsoft.Identity.Web
```

## Startup.cs

In the `Startup.cs` file, the `AddMicrosoftIdentityWebApp` method adds Microsoft.Identity.Web to your web app. The `AddMicrosoftGraph` method adds Microsoft Graph support.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Identity.Web;
using Microsoft.AspNetCore.Authentication.OpenIdConnect;

// Some code omitted for brevity.
public class Startup
{
 // This method gets called by the runtime. Use this method to add services to the container.
 public void ConfigureServices(IServiceCollection services)
 {
 services.AddAuthentication(OpenIdConnectDefaults.AuthenticationScheme)
 .AddMicrosoftIdentityWebApp(Configuration.GetSection("AzureAd"))
 .EnableTokenAcquisitionToCallDownstreamApi()
 .AddMicrosoftGraph(Configuration.GetSection("Graph"))
 .AddInMemoryTokenCaches();

 services.AddRazorPages();
 }
}
```

## appsettings.json

`AzureAd` specifies the configuration for the Microsoft.Identity.Web library. In the [Azure portal](#), select **Azure Active Directory** from the portal menu and then select **App registrations**. Select the app registration created when you enabled the App Service authentication/authorization module. (The app registration should have the same name as your web app.) You can find the tenant ID and client ID in the app registration overview page. The domain name can be found in the Azure AD overview page for your tenant.

`Graph` specifies the Microsoft Graph endpoint and the initial scopes needed by the app.

```
{
 "AzureAd": {
 "Instance": "https://login.microsoftonline.com/",
 "Domain": "fourthcoffeetest.onmicrosoft.com",
 "TenantId": "[tenant-id]",
 "ClientId": "[client-id]",
 // To call an API
 "ClientSecret": "[secret-from-portal]", // Not required by this scenario
 "CallbackPath": "/signin-oidc"
 },

 "Graph": {
 "BaseUrl": "https://graph.microsoft.com/v1.0",
 "Scopes": "user.read"
 },
 "Logging": {
 "LogLevel": {
 "Default": "Information",
 "Microsoft": "Warning",
 "Microsoft.Hosting.Lifetime": "Information"
 }
 },
 "AllowedHosts": "*"
}
```

## Index.cshtml.cs

The following example shows how to call Microsoft Graph as the signed-in user and get some user information. The `GraphServiceClient` object is injected into the controller, and authentication has been configured for you by the Microsoft.Identity.Web library.

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Microsoft.Graph;
using System.IO;
using Microsoft.Identity.Web;
using Microsoft.Extensions.Logging;

// Some code omitted for brevity.

[AuthorizeForScopes(Scopes = new[] { "user.read" })]
public class IndexModel : PageModel
{
 private readonly ILogger<IndexModel> _logger;
 private readonly GraphServiceClient _graphServiceClient;

 public IndexModel(ILogger<IndexModel> logger, GraphServiceClient graphServiceClient)
 {
 _logger = logger;
 _graphServiceClient = graphServiceClient;
 }

 public async Task OnGetAsync()
 {
 try
 {
 var user = await _graphServiceClient.Me.Request().GetAsync();
 ViewData["Me"] = user;
 ViewData["name"] = user.DisplayName;

 using (var photoStream = await _graphServiceClient.Me.Photo.Content.Request().GetAsync())
 {
 byte[] photoByte = ((MemoryStream)photoStream).ToArray();
 ViewData["photo"] = Convert.ToBase64String(photoByte);
 }
 }
 catch (Exception ex)
 {
 ViewData["photo"] = null;
 }
 }
}

```

## Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, [clean up the resources you created](#).

## Next steps

In this tutorial, you learned how to:

- Grant delegated permissions to a web app.
- Call Microsoft Graph from a web app for a signed-in user.

[App service accesses Microsoft Graph as the app](#)

# Tutorial: Access Microsoft Graph from a secured app as the app

11/2/2021 • 5 minutes to read • [Edit Online](#)

Learn how to access Microsoft Graph from a web app running on Azure App Service.

You want to call Microsoft Graph for the web app. A safe way to give your web app access to data is to use a [system-assigned managed identity](#). A managed identity from Azure Active Directory allows App Service to access resources through role-based access control (RBAC), without requiring app credentials. After assigning a managed identity to your web app, Azure takes care of the creation and distribution of a certificate. You don't have to worry about managing secrets or app credentials.

In this tutorial, you learn how to:

- Create a system-assigned managed identity on a web app.
- Add Microsoft Graph API permissions to a managed identity.
- Call Microsoft Graph from a web app by using managed identities.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- A web application running on Azure App Service that has the [App Service authentication/authorization module enabled](#).

## Enable managed identity on app

If you create and publish your web app through Visual Studio, the managed identity was enabled on your app for you. In your app service, select **Identity** in the left pane and then select **System assigned**. Verify that **Status** is set to **On**. If not, select **Save** and then select **Yes** to enable the system-assigned managed identity. When the managed identity is enabled, the status is set to **On** and the object ID is available.

Take note of the **Object ID** value, which you'll need in the next step.

The screenshot shows the Azure portal interface for managing a managed identity. The left sidebar lists several sections: Security, Events (preview), Deployment (with Quickstart, Deployment slots, Deployment Center, and Deployment Center (Preview)), Settings (with Configuration, Authentication / Authorization, Application Insights), and Identity. The Identity section is highlighted with a red box. The main content area shows the current configuration for the managed identity:

- Type:** System assigned (highlighted with a red box).
- Status:** On (highlighted with a red box).
- Object ID:** 09382857-REDACTED-4138df10bfa2 (highlighted with a red box).
- Permissions:** Azure role assignments.

A note at the bottom states: "This resource is registered with Azure Active Directory. You can control it".

## Grant access to Microsoft Graph

When accessing the Microsoft Graph, the managed identity needs to have proper permissions for the operation it wants to perform. Currently, there's no option to assign such permissions through the Azure portal. The following script will add the requested Microsoft Graph API permissions to the managed identity service principal object.

- [PowerShell](#)
- [Azure CLI](#)

```

Install the module. (You need admin on the machine.)
Install-Module AzureAD.

Your tenant ID (in the Azure portal, under Azure Active Directory > Overview).
$TenantID=<tenant-id>
$resourceGroup = "securewebappresourcegroup"
$webAppName="SecureWebApp-2020102125811"

Get the ID of the managed identity for the web app.
$spID = (Get-AzWebApp -ResourceGroupName $resourceGroup -Name $webAppName).identity.principalId

Check the Microsoft Graph documentation for the permission you need for the operation.
$PermissionName = "User.Read.All"

Connect-AzureAD -TenantId $TenantID

Get the service principal for Microsoft Graph.
First result should be AppId 00000003-0000-0000-c000-000000000000
$GraphServicePrincipal = Get-AzureADServicePrincipal -SearchString "Microsoft Graph" | Select-Object -first 1

Assign permissions to the managed identity service principal.
$appRole = $GraphServicePrincipal.AppRoles | `

Where-Object {$_._Value -eq $PermissionName -and $_._AllowedMemberTypes -contains "Application"}`

New-AzureAdServiceAppRoleAssignment -ObjectId $spID -PrincipalId $spID `

-ResourceId $GraphServicePrincipal.ObjectId -Id $appRole.Id

```

After executing the script, you can verify in the [Azure portal](#) that the requested API permissions are assigned to the managed identity.

Go to **Azure Active Directory**, and then select **Enterprise applications**. This pane displays all the service principals in your tenant. In **All Applications**, select the service principal for the managed identity.

If you're following this tutorial, there are two service principals with the same display name (SecureWebApp20200924113531, for example). The service principal that has a **Homepage URL** represents the web app in your tenant. The service principal without the **Homepage URL** represents the system-assigned managed identity for your web app. The **Object ID** value for the managed identity matches the object ID of the managed identity that you previously created.

Select the service principal for the managed identity.

Name	Homepage URL	Object ID	Application ID
SecureWebApp20200924113531		09382857-ed3f-4cbd-b524-4138df10bfa2	1e448214-461a-4efc-b031-454...
SecureWebApp20200924113531	https://securewebapp20200924...	3c52e043-be96-4a9f-b950-d4137eba91de	547082a3-7171-465b-a59d-9c6...

In **Overview**, select **Permissions**, and you'll see the added permissions for Microsoft Graph.

The screenshot shows the 'Permissions' page for an enterprise application named 'SecureWebApp20200924113531'. The left sidebar includes sections for Overview, Deployment Plan, Diagnose and solve problems, Manage (Properties, Owners, Roles and administrators, Users and groups, Single sign-on, Provisioning, Self-service), Security (Permissions, Token encryption), Activity, and Sign-ins. The 'Permissions' section is highlighted with a red box. The main content area displays the 'Permissions' table with one row for 'Microsoft Graph'. The row is also highlighted with a red box. The table columns are API Name, Permission, Type, Granted through, and Granted by.

API Name	Permission	Type	Granted through	Granted by
Microsoft Graph	Read all users' full profil...	Application	Admin consent	An administrator

## Call Microsoft Graph (.NET)

The `DefaultAzureCredential` class is used to get a token credential for your code to authorize requests to Microsoft Graph. Create an instance of the `DefaultAzureCredential` class, which uses the managed identity to fetch tokens and attach them to the service client. The following code example gets the authenticated token credential and uses it to create a service client object, which gets the users in the group.

To see this code as part of a sample application, see the [sample on GitHub](#).

### Install the `Microsoft.Identity.Web.MicrosoftGraph` client library package

Install the [Microsoft.Identity.Web.MicrosoftGraph NuGet package](#) in your project by using the .NET Core command-line interface or the Package Manager Console in Visual Studio.

- [Command line](#)
- [Package Manager](#)

Open a command line, and switch to the directory that contains your project file.

Run the install commands.

```
dotnet add package Microsoft.Identity.Web.MicrosoftGraph
```

### Example

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Azure.Identity;
using Microsoft.Graph.Core;
using System.Net.Http.Headers;

...

public IList<MSGraphUser> Users { get; set; }

public async Task OnGetAsync()
{
 // Create the Microsoft Graph service client with a DefaultAzureCredential class, which gets an access
 // token by using the available Managed Identity.
 var credential = new DefaultAzureCredential();
 var token = credential.GetToken(
 new Azure.Core.TokenRequestContext(
 new[] { "https://graph.microsoft.com/.default" }));

 var accessToken = token.Token;
 var graphServiceClient = new GraphServiceClient(
 new DelegateAuthenticationProvider((requestMessage) =>
 {
 requestMessage
 .Headers
 .Authorization = new AuthenticationHeaderValue("bearer", accessToken);

 return Task.CompletedTask;
 }));
}

List<MSGraphUser> msGraphUsers = new List<MSGraphUser>();
try
{
 var users = await graphServiceClient.Users.Request().GetAsync();
 foreach(var u in users)
 {
 MSGraphUser user = new MSGraphUser();
 user.userPrincipalName = u.UserPrincipalName;
 user.displayName = u.DisplayName;
 user.mail = u.Mail;
 user.jobTitle = u.JobTitle;

 msGraphUsers.Add(user);
 }
}
catch(Exception ex)
{
 string msg = ex.Message;
}

Users = msGraphUsers;
}

```

## Clean up resources

If you're finished with this tutorial and no longer need the web app or associated resources, [clean up the resources you created](#).

## Next steps

In this tutorial, you learned how to:

- Create a system-assigned managed identity on a web app.
- Add Microsoft Graph API permissions to a managed identity.
- Call Microsoft Graph from a web app by using managed identities.

Learn how to connect a [.NET Core app](#), [Python app](#), [Java app](#), or [Node.js app](#) to a database.

# Tutorial: Clean up resources

11/2/2021 • 2 minutes to read • [Edit Online](#)

If you completed all the steps in this multipart tutorial, you created an app service, app service hosting plan, and a storage account in a resource group. You also created an app registration in Azure Active Directory. When no longer needed, delete these resources and app registration so that you don't continue to accrue charges.

In this tutorial, you learn how to:

- Delete the Azure resources created while following the tutorial.

## Delete the resource group

In the [Azure portal](#), select **Resource groups** from the portal menu and select the resource group that contains your app service and app service plan.

Select **Delete resource group** to delete the resource group and all the resources.

The screenshot shows the Azure portal interface. On the left, the 'Resource groups' blade is open, showing a list of resource groups. One resource group, 'SecureWebApp', is selected and highlighted with a red box. On the right, the detailed view for the 'SecureWebApp' resource group is shown. At the top of this view, there is a red box around the 'Delete resource group' button. The main area displays the resource group's configuration, including its subscription (Test Subscription), deployment status (2 Succeeded), and a list of resources it contains. The resources listed are: 'SecureWebApp20200914152557' (App Service, South Central US), 'SecureWebApp20200914152557Plan' (App Service plan, South Central US), and 'securewebappstorage' (Storage account, South Central US).

This command might take several minutes to run.

## Delete the app registration

From the portal menu, select **Azure Active Directory > App registrations**. Then select the application you created.

Home > Fourth Coffee

## Fourth Coffee | App registrations

Azure Active Directory

New registration Endpoints Troubleshooting Download (Preview) Got feedback?

Welcome to the new and improved App registrations (now Generally Available). See what's new and learn more on how it's changed.

Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure AD Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more

All applications Owned applications

Start typing a name or Application ID to filter these results

Display name	Application (client) ID	Created on	Certificates & secrets
SecureWebApp20200914152557	6f88c862-205c-4442-8...	9/14/2020	Current

Manage

- Overview
- Getting started
- Preview hub
- Diagnose and solve problems
- Users
- Groups
- External Identities
- Roles and administrators
- Administrative units (Preview)
- Enterprise applications
- Devices
- App registrations

In the app registration overview, select **Delete**.

Home > Fourth Coffee > SecureWebApp20200914152557

## SecureWebApp20200914152557

Search (Ctrl+ /) Delete Endpoints

Overview Quickstart Integration assistant | Preview Manage

Got a s Delete e would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Essentials

Display name	Supported account types
SecureWebApp20200914152557	My organization only

## Next steps

In this tutorial, you learned how to:

- Delete the Azure resources created while following the tutorial.

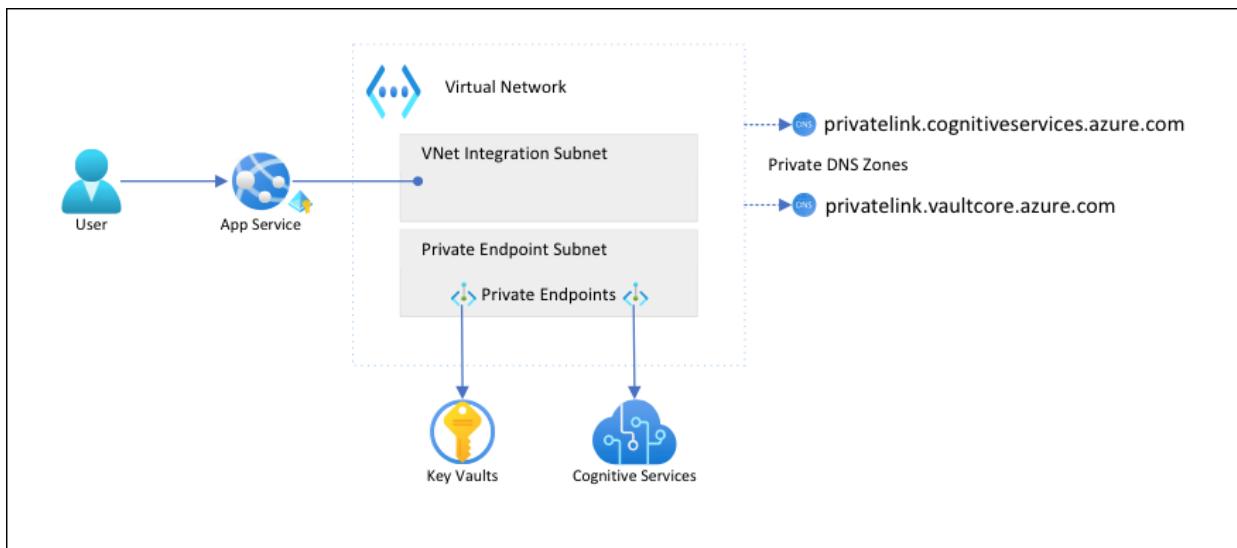
Learn how to connect a [.NET Core app](#), [Python app](#), [Java app](#), or [Nodejs app](#) to a database.

# Tutorial: Isolate back-end communication in Azure App Service with Virtual Network integration

11/2/2021 • 7 minutes to read • [Edit Online](#)

In this article you will configure an App Service app with secure, network-isolated communication to backend services. The example scenario used is in [Tutorial: Secure Cognitive Service connection from App Service using Key Vault](#). When you're finished, you have an App Service app that accesses both Key Vault and Cognitive Services through an [Azure virtual network](#) (VNet), and no other traffic is allowed to access those back-end resources. All traffic will be isolated within your VNet using [VNet integration](#) and [private endpoints](#).

As a multi-tenanted service, outbound network traffic from your App Service app to other Azure services shares the same environment with other apps or even other subscriptions. While the traffic itself can be encrypted, certain scenarios may require an extra level of security by isolating back-end communication from other network traffic. These scenarios are typically accessible to large enterprises with a high level of expertise, but App Service puts it within reach with VNet integration.



With this architecture:

- Public traffic to the back-end services is blocked.
- Outbound traffic from App Service is routed to the VNet and can reach the back-end services.
- App Service is able to perform DNS resolution to the back-end services through the private DNS zones.

What you will learn:

- Create a VNet and subnets for App Service VNet integration
- Create private DNS zones
- Create private endpoints
- Configure VNet integration in App Service

## Prerequisites

The tutorial assumes that you have followed the [Tutorial: Secure Cognitive Service connection from App Service using Key Vault](#) and created the language detector app.

The tutorial continues to use the following environment variables from the previous tutorial. Make sure you set

them properly.

```
groupName=myKVResourceGroup
region=westeurope
csResourceName=<cs-resource-name>
appName=<app-name>
vaultName=<vault-name>
```

## Create VNet and subnets

1. Create a VNet. Replace <*virtual-network-name*> with a unique name.

```
Save vnet name as variable for convenience
vnetName=<virtual-network-name>

az network vnet create --resource-group $groupName --location $region --name $vnetName --address-
prefixes 10.0.0.0/16
```

2. Create a subnet for the App Service VNet integration.

```
az network vnet subnet create --resource-group $groupName --vnet-name $vnetName --name vnet-
integration-subnet --address-prefixes 10.0.0.0/24 --delegations Microsoft.Web/serverfarms
```

For App Service, the VNet integration subnet is recommended to have a CIDR block of /26 at a minimum (see [VNet integration subnet requirements](#)). /24 is more than sufficient.

--delegations Microsoft.Web/serverfarms specifies that the subnet is [delegated for App Service VNet integration](#).

3. Create another subnet for the private endpoints.

```
az network vnet subnet create --resource-group $groupName --vnet-name $vnetName --name private-
endpoint-subnet --address-prefixes 10.0.1.0/24 --disable-private-endpoint-network-policies
```

For private endpoint subnets, you must [disable private endpoint network policies](#).

## Create private DNS zones

Because your Key Vault and Cognitive Services resources will sit behind [private endpoints](#), you need to define [private DNS zones](#) for them. These zones are used to host the DNS records for private endpoints and allow the clients to find the back-end services by name.

1. Create two private DNS zones, one for your Cognitive Services resource and one for your key vault.

```
az network private-dns zone create --resource-group $groupName --name
privatelink.cognitiveservices.azure.com
az network private-dns zone create --resource-group $groupName --name privatelink.vaultcore.azure.net
```

For more information on these settings, see [Azure Private Endpoint DNS configuration](#)

2. Link the private DNS zones to the VNet.

```
az network private-dns link vnet create --resource-group $groupName --name cognitiveservices-zonelink
--zone-name privatelink.cognitiveservices.azure.com --virtual-network $vnetName --registration-enabled False
az network private-dns link vnet create --resource-group $groupName --name vaultcore-zonelink --zone-name privatelink.vaultcore.azure.net --virtual-network $vnetName --registration-enabled False
```

## Create private endpoints

1. In the private endpoint subnet of your VNet, create a private endpoint for your key vault.

```
Get Cognitive Services resource ID
csResourceId=$(az cognitiveservices account show --resource-group $groupName --name $csResourceName --query id --output tsv)

az network private-endpoint create --resource-group $groupName --name securecstext-pe --location $region --connection-name securecstext-pc --private-connection-resource-id $csResourceId --group-id account --vnet-name $vnetName --subnet private-endpoint-subnet
```

2. Create a DNS zone group for the Cognitive Services private endpoint. DNS zone group is a link between the private DNS zone and the private endpoint. This link helps you to auto update the private DNS Zone when there is an update to the private endpoint.

```
az network private-endpoint dns-zone-group create --resource-group $groupName --endpoint-name securecstext-pe --name securecstext-zg --private-dns-zone privatelink.cognitiveservices.azure.com --zone-name privatelink.cognitiveservices.azure.com
```

3. Block public traffic to the Cognitive Services resource.

```
az rest --uri $csResourceId?api-version=2021-04-30 --method PATCH --body '{"properties": {"publicNetworkAccess": "Disabled"}}' --headers 'Content-Type=application/json'

Repeat following command until output is "Succeeded"
az cognitiveservices account show --resource-group $groupName --name $csResourceName --query properties.provisioningState
```

### NOTE

Make sure the provisioning state of your change is "Succeeded". Then you can observe the behavior change in the sample app. You can still load the app, but if you try click the Detect button, you get an HTTP 500 error. The app has lost its connectivity to the Cognitive Services resource through the shared networking.

4. Repeat the steps above for the key vault.

```
Create private endpoint for key vault
vaultResourceId=$(az keyvault show --name $vaultName --query id --output tsv)
az network private-endpoint create --resource-group $groupName --name securekeyvault-pe --location $region --connection-name securekeyvault-pc --private-connection-resource-id $vaultResourceId --group-id vault --vnet-name $vnetName --subnet private-endpoint-subnet
Create DNS zone group for the endpoint
az network private-endpoint dns-zone-group create --resource-group $groupName --endpoint-name securekeyvault-pe --name securekeyvault-zg --private-dns-zone privatelink.vaultcore.azure.net --zone-name privatelink.vaultcore.azure.net
Block public traffic to key vault
az keyvault update --name $vaultName --default-action Deny
```

- Force an immediate refetch of the [key vault references](#) in your app by resetting the app settings (for more information, see [Rotation](#)).

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings
CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)"
CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

**NOTE**

Again, you can observe the behavior change in the sample app. You can no longer load the app because it can no longer access the key vault references. The app has lost its connectivity to the key vault through the shared networking.

The two private endpoints are only accessible to clients inside the VNet you created. You can't even access the secrets in the key vault through [Secrets](#) page in the Azure portal, because the portal accesses them through the public internet (see [Manage the locked down resources](#)).

## Configure VNet integration in your app

- Scale the app up to **Standard** tier. VNet integration requires **Standard** tier or above (see [Integrate your app with an Azure virtual network](#)).

```
az appservice plan update --name $appName --resource-group $groupName --sku S1
```

- Unrelated to our scenario but also important, enforce HTTPS for inbound requests.

```
az webapp update --resource-group $groupName --name $appName --https-only
```

- Enable VNet integration on your app.

```
az webapp vnet-integration add --resource-group $groupName --name $appName --vnet $vnetName --subnet
vnet-integration-subnet
```

VNet integration allows outbound traffic to flow directly into the VNet. By default, only local IP traffic defined in [RFC-1918](#) is routed to the VNet, which is what you need for the private endpoints. To route all your traffic to the VNet, see [Manage virtual network integration routing](#). Routing all traffic can also be used if you want to route internet traffic through your VNet, such as through an [Azure VNet NAT](#) or an [Azure Firewall](#).

- In the browser, navigate to `<app-name>.azurewebsites.net` again and wait for the integration to take effect. If you get an HTTP 500 error, wait a few minutes and try again. If you can load the page and get detection results, then you're connecting to the Cognitive Services endpoint with key vault references.

**NOTE**

If keep getting HTTP 500 errors after a long time, it may help to force a refetch of the [key vault references](#) again, like so:

```
az webapp config appsettings set --resource-group $groupName --name $appName --settings
CS_ACCOUNT_NAME="@Microsoft.KeyVault(SecretUri=$csResourceKVUri)"
CS_ACCOUNT_KEY="@Microsoft.KeyVault(SecretUri=$csKeyKVUri)"
```

## Manage the locked down resources

Depending on your scenarios, you may not be able to manage the private endpoint protected resources through the Azure portal, Azure CLI, or Azure PowerShell (for example, Key Vault). These tools all make REST API calls to access the resources through the public internet, and are blocked by your configuration. Here are a few options for accessing the locked down resources:

- For Key Vault, add the public IP of your local machine to view or update the private endpoint protected secrets.
- If your on premises network is extended into the Azure VNet through a [VPN gateway](#) or [ExpressRoute](#), you can manage the private endpoint protected resources directly from your on premises network.
- Manage the private endpoint protected resources from a [jump server](#) in the VNet.
- [Deploy Cloud Shell into the VNet](#).

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name $groupName
```

This command may take a minute to run.

## Next steps

- [Integrate your app with an Azure virtual network](#)
- [App Service networking features](#)

# Tutorial: Host a RESTful API with CORS in Azure App Service

11/2/2021 • 9 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. In addition, App Service has built-in support for [Cross-Origin Resource Sharing \(CORS\)](#) for RESTful APIs. This tutorial shows how to deploy an ASP.NET Core API app to App Service with CORS support. You configure the app using command-line tools and deploy the app using Git.

In this tutorial, you learn how to:

- Create App Service resources using Azure CLI
- Deploy a RESTful API to Azure using Git
- Enable App Service CORS support

You can follow the steps in this tutorial on macOS, Linux, Windows.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install the latest .NET Core 3.1 SDK](#)

## Create local ASP.NET Core app

In this step, you set up the local ASP.NET Core project. App Service supports the same workflow for APIs written in other languages.

### Clone the sample application

1. In the terminal window, `cd` to a working directory.
2. Clone the sample repository and change to the repository root.

```
git clone https://github.com/Azure-Samples/dotnet-core-api
cd dotnet-core-api
```

This repository contains an app that's created based on the following tutorial: [ASP.NET Core Web API help pages using Swagger](#). It uses a Swagger generator to serve the [Swagger UI](#) and the Swagger JSON endpoint.

3. Make sure the default branch is `main`.

```
git branch -m main
```

### TIP

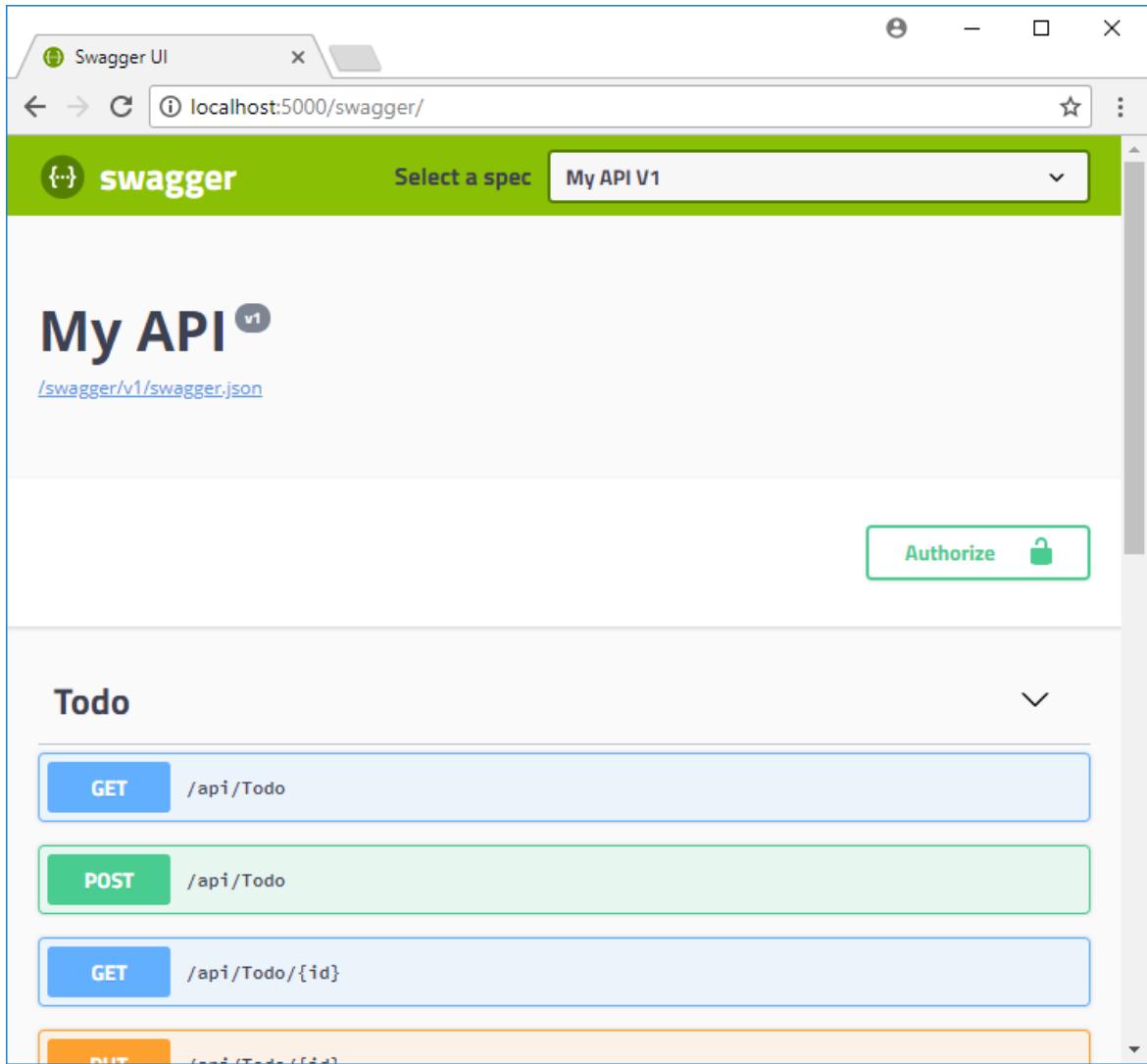
The branch name change isn't required by App Service. However, since many repositories are changing their default branch to `main` (see [Change deployment branch](#)), this tutorial also shows you how to deploy a repository from `main`.

## Run the application

1. Run the following commands to install the required packages, run database migrations, and start the application.

```
dotnet restore
dotnet run
```

2. Navigate to `http://localhost:5000/swagger` in a browser to play with the Swagger UI.



3. Navigate to `http://localhost:5000/api/todo` and see a list of ToDo JSON items.
4. Navigate to `http://localhost:5000` and play with the browser app. Later, you will point the browser app to a remote API in App Service to test CORS functionality. Code for the browser app is found in the repository's `wwwroot` directory.
5. To stop ASP.NET Core at any time, press `ctrl+c` in the terminal.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

## Deploy app to Azure

In this step, you deploy your SQL Database-connected .NET Core application to App Service.

### Configure local git deployment

FTP and local Git can deploy to an Azure web app by using a *deployment user*. Once you configure your deployment user, you can use it for all your Azure deployments. Your account-level deployment username and password are different from your Azure subscription credentials.

To configure the deployment user, run the [az webapp deployment user set](#) command in Azure Cloud Shell.

Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

Record your username and password to use to deploy your web apps.

### Create a resource group

A [resource group](#) is a logical container into which Azure resources, such as web apps, databases, and storage accounts, are deployed and managed. For example, you can choose to delete the entire resource group in one

simple step later.

In the Cloud Shell, create a resource group with the `az group create` command. The following example creates a resource group named `myResourceGroup` in the *West Europe* location. To see all supported locations for App Service in **Free** tier, run the `az appservice list-locations --sku FREE` command.

```
az group create --name myResourceGroup --location "West Europe"
```

You generally create your resource group and the resources in a region near you.

When the command finishes, a JSON output shows you the resource group properties.

### Create an App Service plan

In the Cloud Shell, create an App Service plan with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier:

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku FREE
```

When the App Service plan has been created, the Azure CLI shows information similar to the following example:

```
{
 "adminSiteName": null,
 "appServicePlanName": "myAppServicePlan",
 "geoRegion": "West Europe",
 "hostingEnvironmentProfile": null,
 "id": "/subscriptions/0000-0000/resourceGroups/myResourceGroup/providers/Microsoft.Web/serverfarms/myAppServicePlan",
 "kind": "app",
 "location": "West Europe",
 "maximumNumberOfWorkers": 1,
 "name": "myAppServicePlan",
 < JSON data removed for brevity. >
 "targetWorkerSizeId": 0,
 "type": "Microsoft.Web/serverfarms",
 "workerTierName": null
}
```

### Create a web app

Create a [web app](#) in the `myAppServicePlan` App Service plan.

In the Cloud Shell, you can use the `az webapp create` command. In the following example, replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --deployment-local-git
```

When the web app has been created, the Azure CLI shows output similar to the following example:

```
Local git is configured with url of 'https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git'
{
 "availabilityState": "Normal",
 "clientAffinityEnabled": true,
 "clientCertEnabled": false,
 "clientCertExclusionPaths": null,
 "cloningInfo": null,
 "containerSize": 0,
 "dailyMemoryTimeQuota": 0,
 "defaultHostName": "<app-name>.azurewebsites.net",
 "deploymentLocalGitUrl": "https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git",
 "enabled": true,
 < JSON data removed for brevity. >
}
```

#### NOTE

The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

`https://<username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Save this URL as you need it later.

### Push to Azure from Git

1. Since you're deploying the `main` branch, you need to set the default deployment branch for your App Service app to `main` (see [Change deployment branch](#)). In the Cloud Shell, set the `DEPLOYMENT_BRANCH` app setting with the `az webapp config appsettings set` command.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
DEPLOYMENT_BRANCH='main'
```

2. Back in the local terminal window, add an Azure remote to your local Git repository. Replace `<deploymentLocalGitUrl-from-create-step>` with the URL of the Git remote that you saved from [Create a web app](#).

```
git remote add azure <deploymentLocalGitUrl-from-create-step>
```

3. Push to the Azure remote to deploy your app with the following command. When Git Credential Manager prompts you for credentials, make sure you enter the credentials you created in [Configure a deployment user](#), not the credentials you use to sign in to the Azure portal.

```
git push azure main
```

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

```
Enumerating objects: 83, done.
Counting objects: 100% (83/83), done.
Delta compression using up to 8 threads
Compressing objects: 100% (78/78), done.
Writing objects: 100% (83/83), 22.15 KiB | 3.69 MiB/s, done.
Total 83 (delta 26), reused 0 (delta 0)
remote: Updating branch 'master'.
remote: Updating submodules.
remote: Preparing deployment for commit id '509236e13d'.
remote: Generating deployment script.
remote: Project file path: .\TodoApi.csproj
remote: Generating deployment script for ASP.NET MSBuild16 App
remote: Generated deployment script files
remote: Running deployment command...
remote: Handling ASP.NET Core Web Application deployment with MSBuild16.
remote: .
remote: .
remote: .
remote: Finished successfully.
remote: Running post deployment command(s)...
remote: Triggering recycle (preview mode disabled).
remote: Deployment successful.
To https://<app_name>.scm.azurewebsites.net/<app_name>.git
 * [new branch] master -> master
```

### Browse to the Azure app

1. Navigate to `http://<app_name>.azurewebsites.net/swagger` in a browser and play with the Swagger UI.

The screenshot shows the Swagger UI interface for a deployed .NET Core API. At the top, it says "My API" with a "v1" badge. Below that is a link to "/swagger/v1/swagger.json". On the right, there's a "Select a spec" dropdown set to "My API V1" and an "Authorize" button with a lock icon. The main area is titled "Todo" with a dropdown arrow. It lists four API operations:

- GET /api/Todo
- POST /api/Todo
- GET /api/Todo/{id}
- PUT /api/Todo/{id}

2. Navigate to `http://<app_name>.azurewebsites.net/swagger/v1/swagger.json` to see the `swagger.json` for your deployed API.
3. Navigate to `http://<app_name>.azurewebsites.net/api/todo` to see your deployed API working.

## Add CORS functionality

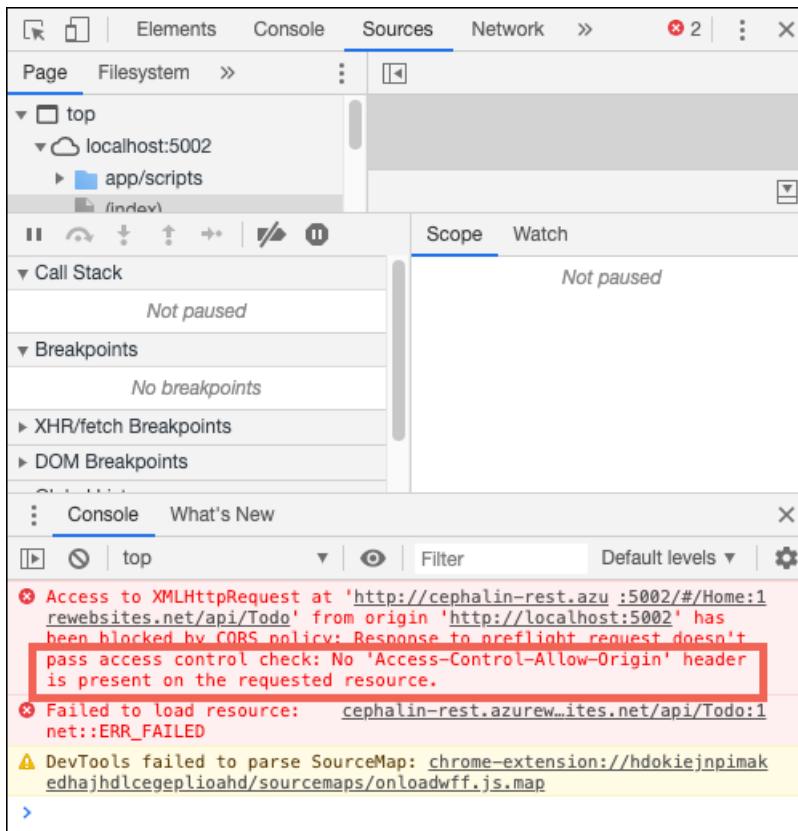
Next, you enable the built-in CORS support in App Service for your API.

### Test CORS in sample app

1. In your local repository, open `wwwroot/index.html`.
2. In Line 51, set the `apiEndpoint` variable to the URL of your deployed API (`http://<app_name>.azurewebsites.net`). Replace `<appname>` with your app name in App Service.
3. In your local terminal window, run the sample app again.

```
dotnet run
```

4. Navigate to the browser app at `http://localhost:5000`. Open the developer tools window in your browser (`Ctrl + Shift + i` in Chrome for Windows) and inspect the **Console** tab. You should now see the error message, `No 'Access-Control-Allow-Origin' header is present on the requested resource`.



Because of the domain mismatch between the browser app (`http://localhost:5000`) and remote resource (`http://<app_name>.azurewebsites.net`), and the fact that your API in App Service is not sending the `Access-Control-Allow-Origin` header, your browser has prevented cross-domain content from loading in your browser app.

In production, your browser app would have a public URL instead of the localhost URL, but the way to enable CORS to a localhost URL is the same as a public URL.

## Enable CORS

In the Cloud Shell, enable CORS to your client's URL by using the `az webapp cors add` command. Replace the `<app-name>` placeholder.

```
az webapp cors add --resource-group myResourceGroup --name <app-name> --allowed-origins
'http://localhost:5000'
```

You can set more than one client URL in `properties.cors.allowedOrigins` (`["URL1", "URL2", ...]`). You can also enable all client URLs with `"['*']"`.

### NOTE

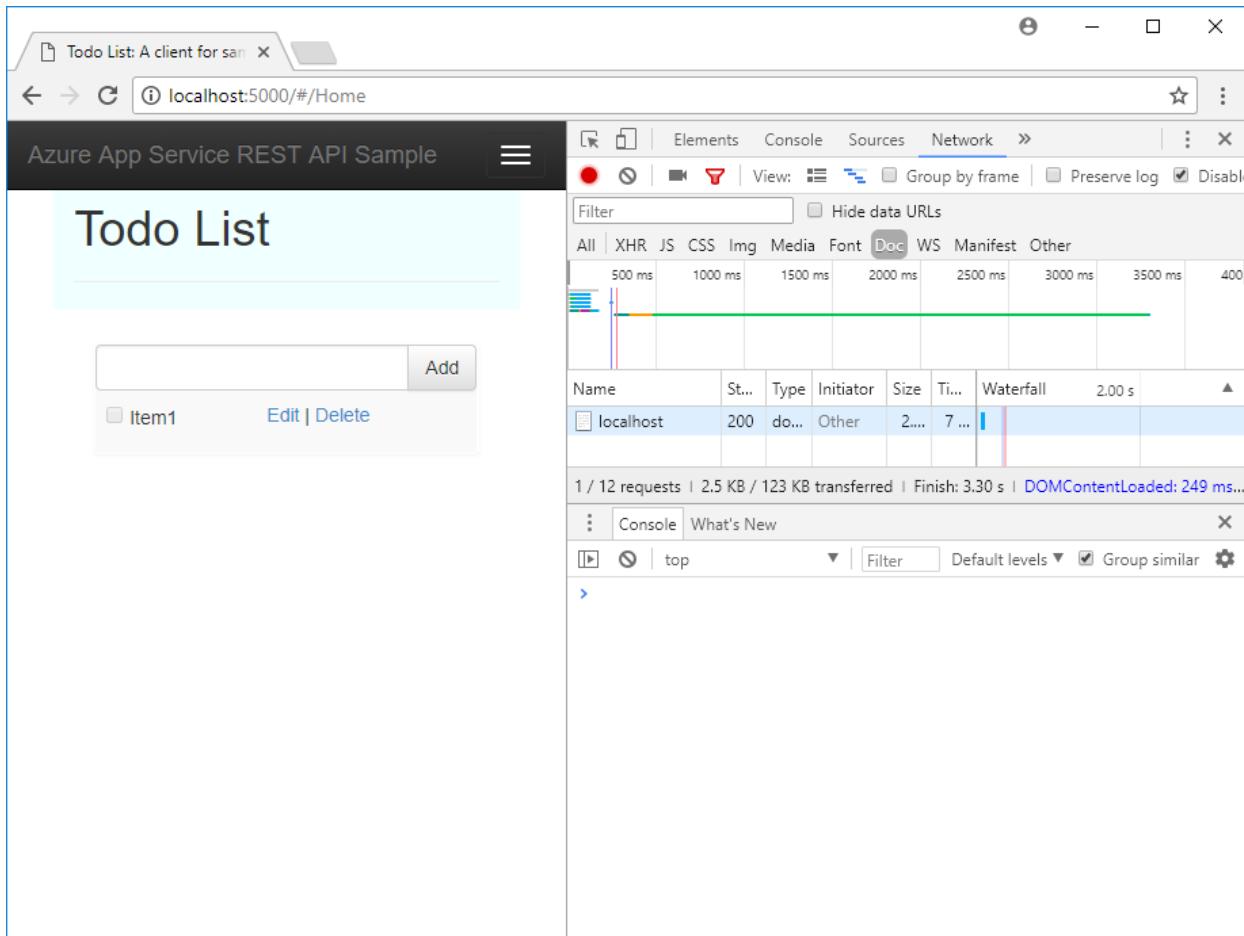
If your app requires credentials such as cookies or authentication tokens to be sent, the browser may require the `ACCESS-CONTROL-ALLOW-CREDENTIALS` header on the response. To enable this in App Service, set `properties.cors.supportCredentials` to `true` in your CORS config. This cannot be enabled when `allowedOrigins` includes `'*'`.

### NOTE

Specifying `AllowAnyOrigin` and `AllowCredentials` is an insecure configuration and can result in cross-site request forgery. The CORS service returns an invalid CORS response when an app is configured with both methods.

## Test CORS again

Refresh the browser app at <http://localhost:5000>. The error message in the **Console** window is now gone, and you can see the data from the deployed API and interact with it. Your remote API now supports CORS to your browser app running locally.



Congratulations, you're running an API in Azure App Service with CORS support.

## App Service CORS vs. your CORS

You can use your own CORS utilities instead of App Service CORS for more flexibility. For example, you may want to specify different allowed origins for different routes or methods. Since App Service CORS lets you specify one set of accepted origins for all API routes and methods, you would want to use your own CORS code. See how ASP.NET Core does it at [Enabling Cross-Origin Requests \(CORS\)](#).

### NOTE

Don't try to use App Service CORS and your own CORS code together. When used together, App Service CORS takes precedence and your own CORS code has no effect.

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

What you learned:

- Create App Service resources using Azure CLI
- Deploy a RESTful API to Azure using Git
- Enable App Service CORS support

Advance to the next tutorial to learn how to authenticate and authorize users.

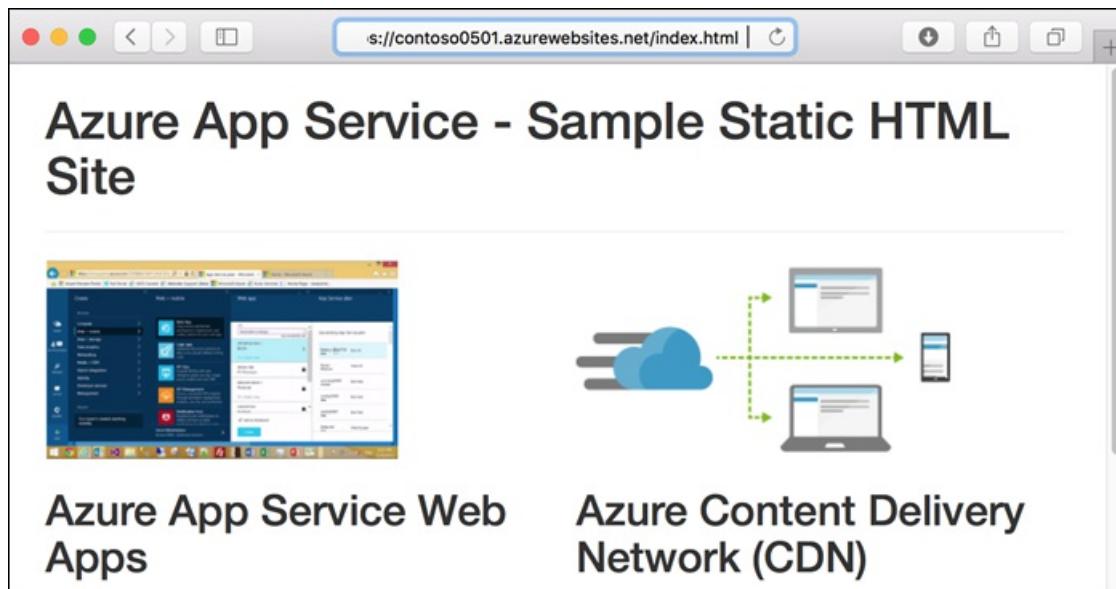
[Tutorial: Authenticate and authorize users end-to-end](#)

# Tutorial: Add Azure CDN to an Azure App Service web app

11/2/2021 • 6 minutes to read • [Edit Online](#)

This tutorial shows how to add [Azure Content Delivery Network \(CDN\)](#) to a [web app](#) in [Azure App Service](#). Web apps is a service for hosting web applications, REST APIs, and mobile back ends.

Here's the home page of the sample static HTML site that you'll work with:



What you'll learn:

- Create a CDN endpoint.
- Refresh cached assets.
- Use query strings to control cached versions.

## Prerequisites

To complete this tutorial:

- [Install Git](#)
- [Install the Azure CLI](#)

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create the web app

To create the web app that you'll work with, follow the [static HTML quickstart](#) through the **Browse to the app** step.

## Log in to the Azure portal

Open a browser and navigate to the [Azure portal](#).

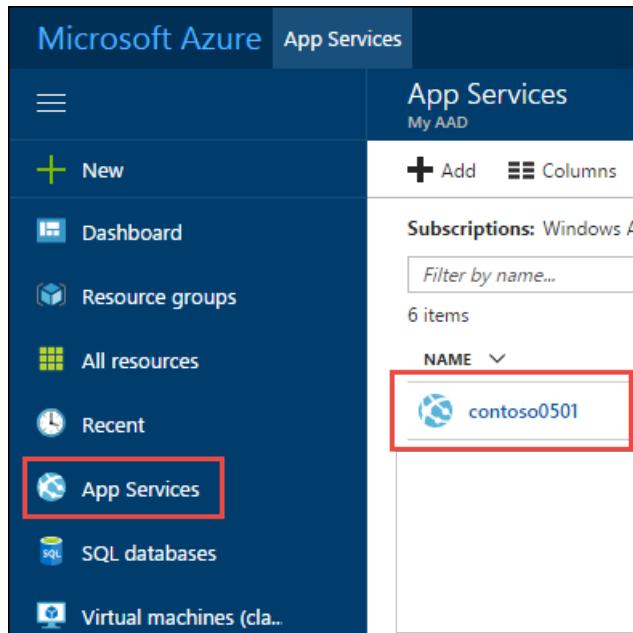
### Dynamic site acceleration optimization

If you want to optimize your CDN endpoint for dynamic site acceleration (DSA), you should use the [CDN portal](#)

to create your profile and endpoint. With [DSA optimization](#), the performance of web pages with dynamic content is measurably improved. For instructions about how to optimize a CDN endpoint for DSA from the CDN portal, see [CDN endpoint configuration to accelerate delivery of dynamic files](#). Otherwise, if you don't want to optimize your new endpoint, you can use the web app portal to create it by following the steps in the next section. Note that for [Azure CDN from Verizon](#) profiles, you cannot change the optimization of a CDN endpoint after it has been created.

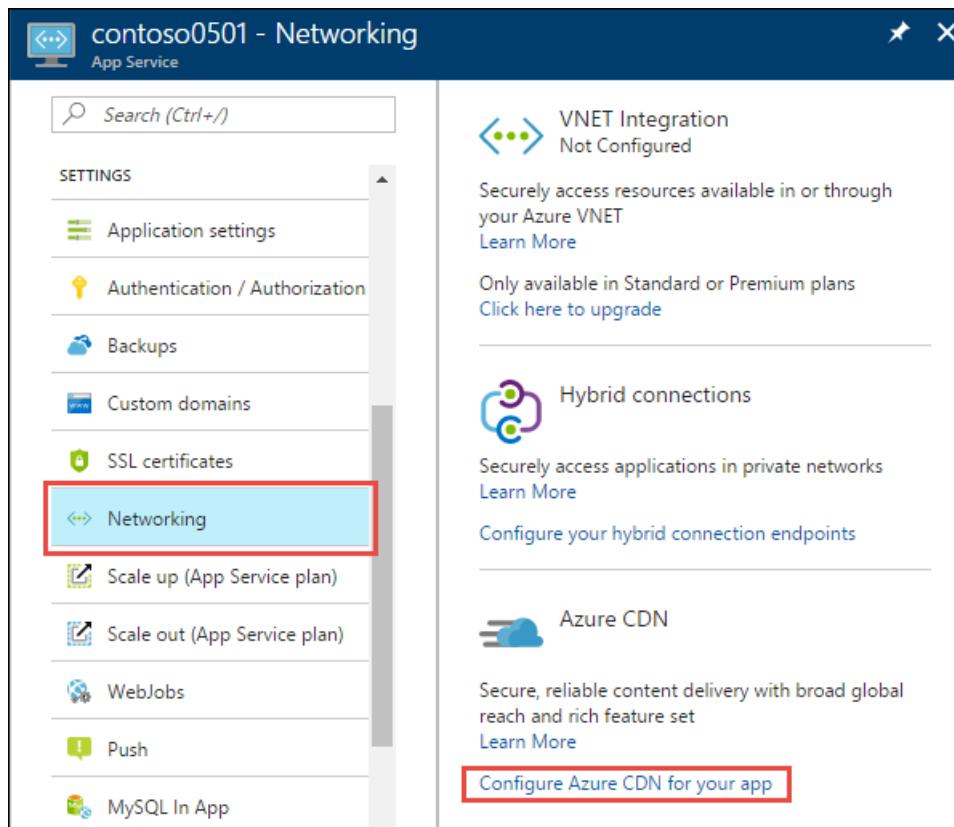
## Create a CDN profile and endpoint

In the left navigation, select **App Services**, and then select the app that you created in the [static HTML quickstart](#).



The screenshot shows the Microsoft Azure portal's App Services dashboard. On the left sidebar, under the 'App Services' category, the 'contoso0501' app is listed and highlighted with a red box. The main pane displays the app's details, including its name and a 'NAME' column header. A 'Filter by name...' search bar is also visible.

In the App Service page, in the **Settings** section, select **Networking > Configure Azure CDN for your app**.



The screenshot shows the 'Networking' settings for the 'contoso0501' app service. The left sidebar lists various settings like Application settings, Authentication / Authorization, and SSL certificates, with 'Networking' highlighted and a red box around it. The right pane shows three sections: 'VNET Integration' (Not Configured), 'Hybrid connections', and 'Azure CDN'. Under 'Azure CDN', there is a link 'Configure Azure CDN for your app' which is also highlighted with a red box.

In the Azure Content Delivery Network page, provide the **New endpoint** settings as specified in the table.

The screenshot shows the 'Azure CDN' blade with the 'Endpoints' section selected. A message states there are no endpoints linked to the resource. Below it, a 'New endpoint' form is displayed. The 'CDN profile' dropdown is set to 'Create new' (selected) and contains the value 'myCDNProfile'. The 'Pricing tier' dropdown is set to 'Standard Akamai'. The 'CDN endpoint name' input field contains 'contoso0501'. The 'Origin hostname' input field contains 'contoso0501.azurewebsites.net'. A 'Create' button is at the bottom.

SETTING	SUGGESTED VALUE	DESCRIPTION
CDN profile	myCDNProfile	A CDN profile is a collection of CDN endpoints with the same pricing tier.
Pricing tier	Standard Akamai	The <a href="#">pricing tier</a> specifies the provider and available features. This tutorial uses <i>Standard Akamai</i> .
CDN endpoint name	Any name that is unique in the azureedge.net domain	You access your cached resources at the domain <endpointname>.azureedge.net.

Select **Create** to create a CDN profile.

Azure creates the profile and endpoint. The new endpoint appears in the **Endpoints** list, and when it's provisioned, the status is **Running**.

Azure CDN

Azure Content Delivery Network

The Azure Content Delivery Network (CDN) is designed to send audio, video, images, and other files faster and more reliably to customers using servers that are closest to the users. This dramatically increases speed and availability, resulting in significant user experience improvements. [Learn more](#)

**Endpoints**

HOSTNAME	STATUS	PROTOCOL	...
contoso0501.azureedge.net	Running	HTTP, HTTPS	...

## Test the CDN endpoint

Because it takes time for the registration to propagate, the endpoint isn't immediately available for use:

- For **Azure CDN Standard from Microsoft profiles**, propagation usually completes in 10 minutes.
- For **Azure CDN Standard from Akamai profiles**, propagation usually completes within one minute.
- For **Azure CDN Standard from Verizon and Azure CDN Premium from Verizon profiles**, propagation usually completes within 90 minutes.

The sample app has an *index.html* file and *css*, *img*, and *js* folders that contain other static assets. The content paths for all of these files are the same at the CDN endpoint. For example, both of the following URLs access the *bootstrap.css* file in the *css* folder:

```
http://<appname>.azurewebsites.net/css/bootstrap.css
```

```
http://<endpointname>.azureedge.net/css/bootstrap.css
```

Navigate a browser to the following URL:

```
http://<endpointname>.azureedge.net/index.html
```

Azure App Service - Sample Static HTML Site

Azure App Service Web Apps

Azure Content Delivery Network (CDN)

You see the same page that you ran earlier in an Azure web app. Azure CDN has retrieved the origin web app's assets and is serving them from the CDN endpoint

To ensure that this page is cached in the CDN, refresh the page. Two requests for the same asset are sometimes required for the CDN to cache the requested content.

For more information about creating Azure CDN profiles and endpoints, see [Getting started with Azure CDN](#).

## Purge the CDN

The CDN periodically refreshes its resources from the origin web app based on the time-to-live (TTL) configuration. The default TTL is seven days.

At times you might need to refresh the CDN before the TTL expiration; for example, when you deploy updated content to the web app. To trigger a refresh, manually purge the CDN resources.

In this section of the tutorial, you deploy a change to the web app and purge the CDN to trigger the CDN to refresh its cache.

### Deploy a change to the web app

Open the *index.htm* file and add - V2 to the H1 heading, as shown in the following example:

```
<h1>Azure App Service - Sample Static HTML Site - V2</h1>
```

Commit your change and deploy it to the web app.

```
git commit -am "version 2"
git push azure main
```

Once deployment has completed, browse to the web app URL to see the change.

```
http://<appname>.azurewebsites.net/index.html
```



If you browse to the CDN endpoint URL for the home page, you won't see the change because the cached version in the CDN hasn't expired yet.

```
http://<endpointname>.azureedge.net/index.html
```



### Purge the CDN in the portal

To trigger the CDN to update its cached version, purge the CDN.

In the portal left navigation, select **Resource groups**, and then select the resource group that you created for your web app (*myResourceGroup*).

The screenshot shows the Microsoft Azure portal's 'Resource groups' page. On the left, there's a sidebar with options like 'New', 'Dashboard', 'Resource groups' (which is selected and highlighted with a red box), 'All resources', and 'Recent'. The main area is titled 'Resource groups' and shows a list of 7 items under 'Subscriptions: Windows Azure MSDN - Visual Studio Ultimate'. A search bar 'Filter by name...' is at the top. The list includes columns for 'NAME' and 'TYPE'. One item, 'myResourceGroup', is highlighted with a red box.

In the list of resources, select your CDN endpoint.

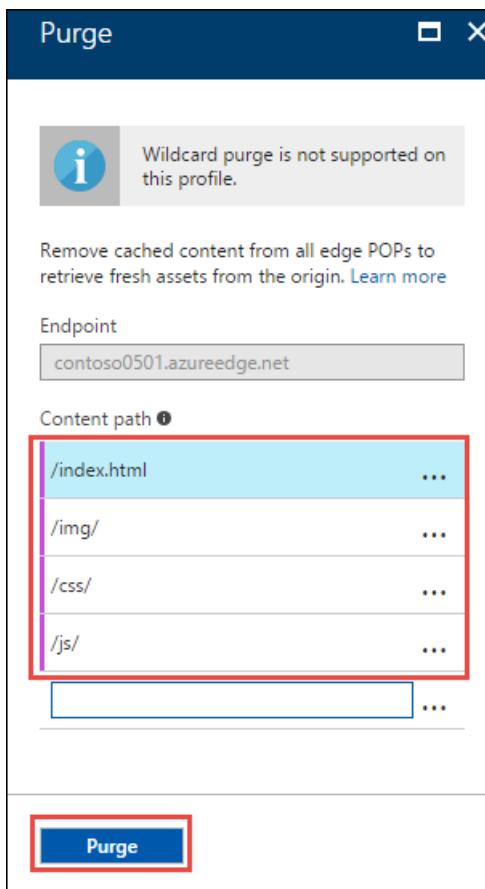
The screenshot shows the 'myResourceGroup' Resource group page. The left sidebar has tabs for 'Overview' (selected and highlighted with a red box), 'Activity log', 'Access control (IAM)', 'Tags', 'SETTINGS' (with 'Quickstart', 'Resource costs', 'Deployments', and 'Properties' sub-options), and 'Essentials' (with 'Subscription name (change)' showing 'Windows Azure MSDN - Visual Studio Ultim...', 'Deployments' showing '1 Succeeded', and a 'Filter by name...' search bar). The main area lists resources: 'contoso0501' (App Service), 'contosocdnprofile' (CDN profile), 'contoso0501' (Endpoint, highlighted with a red box), and 'quickStartPlan' (App Service plan).

At the top of the Endpoint page, select Purge.

The screenshot shows the 'contoso0501' Endpoint page. The left sidebar has tabs for 'Overview' (selected and highlighted with a red box), 'Activity log', and 'Access control (IAM)'. The top navigation bar includes 'Custom domain', 'Purge' (highlighted with a red box), 'Load', and 'Stop'. The 'Essentials' section shows 'Resource group: myResourceGroup' and 'Status: Running'.

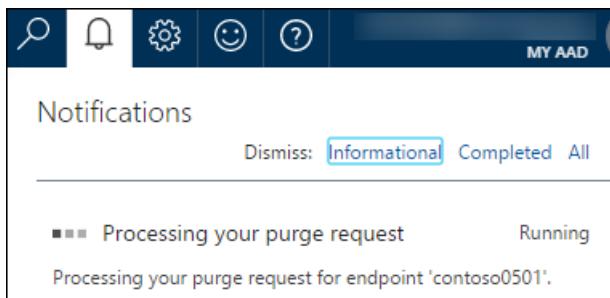
Enter the content paths you want to purge. You can pass a complete file path to purge an individual file, or a path segment to purge and refresh all content in a folder. Because you changed *index.html*, ensure that is in one of the paths.

At the bottom of the page, select Purge.



### Verify that the CDN is updated

Wait until the purge request finishes processing, which is typically a couple of minutes. To see the current status, select the bell icon at the top of the page.



When you browse to the CDN endpoint URL for *index.html*, you'll see the *V2* that you added to the title on the home page, which indicates that the CDN cache has been refreshed.

A screenshot of a web browser window. The address bar shows "https://contoso0501.azureedge.net/index.html". The page content is "Azure App Service - Sample Static HTML Site - V2".

For more information, see [Purge an Azure CDN endpoint](#).

## Use query strings to version content

Azure CDN offers the following caching behavior options:

- Ignore query strings

- Bypass caching for query strings
- Cache every unique URL

The first option is the default, which means there is only one cached version of an asset regardless of the query string in the URL.

In this section of the tutorial, you change the caching behavior to cache every unique URL.

### Change the cache behavior

In the Azure portal CDN Endpoint page, select Cache.

Select Cache every unique URL from the Query string caching behavior drop-down list.

Select Save.

contoso0501 - Cache  
Endpoint

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Origin

Custom domains

Compression

**Cache**

Save Discard

About This Feature

Decide how CDN caches requests that include query strings. You can ignore any query contain query strings from being cached, or cache every request with a unique URL.

Learn more

Configure

Query string caching behavior

- Ignore query strings
- Ignore query strings
- Bypass caching for query strings
- Cache every unique URL**

### Verify that unique URLs are cached separately

In a browser, navigate to the home page at the CDN endpoint, and include a query string:

```
http://<endpointname>.azureedge.net/index.html?q=1
```

Azure CDN returns the current web app content, which includes V2 in the heading.

To ensure that this page is cached in the CDN, refresh the page.

Open *index.html*, change V2 to V3, then deploy the change.

```
git commit -am "version 3"
git push azure main
```

In a browser, go to the CDN endpoint URL with a new query string, such as `q=2`. Azure CDN gets the current *index.html* file and displays V3. However, if you navigate to the CDN endpoint with the `q=1` query string, you see V2.

```
http://<endpointname>.azureedge.net/index.html?q=2
```



```
http://<endpointname>.azureedge.net/index.html?q=1
```



This output shows that each query string is treated differently:

- q=1 was used before, so cached contents are returned (V2).
- q=2 is new, so the latest web app contents are retrieved and returned (V3).

For more information, see [Control Azure CDN caching behavior with query strings](#).

## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

## Next steps

What you learned:

- Create a CDN endpoint.
- Refresh cached assets.
- Use query strings to control cached versions.

Learn how to optimize CDN performance in the following articles:

[Tutorial: Add a custom domain to your Azure CDN endpoint](#)

# Tutorial: Send email and invoke other business processes from App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

In this tutorial, you learn how to integrate your App Service app with your business processes. This is common to web app scenarios, such as:

- Send confirmation email for a transaction
- Add user to Facebook group
- Connect to third-party systems like SAP, Salesforce, etc.
- Exchange standard B2B messages

In this tutorial, you send emails with Gmail from your App Service app by using [Azure Logic Apps](#). There are other ways to send emails from a web app, such as SMTP configuration provided by your language framework. However, Logic Apps brings a lot more power to your App Service app without adding complexity to your code. Logic Apps provides a simple configuration interface for the most popular business integrations, and your app can call them anytime with an HTTP request.

## Prerequisite

Deploy an app with the language framework of your choice to App Service. To follow a tutorial to deploy a sample app, see below:

- [ASP.NET](#)
- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)

[Tutorial: Build an ASP.NET app in Azure with SQL Database](#)

## Create the logic app

1. In the [Azure portal](#), create an empty logic app by following the instructions in [Create your first logic app](#). When you see the **Logic Apps Designer**, return to this tutorial.
2. In the splash page for Logic Apps Designer, select **When an HTTP request is received** under **Start with a common trigger**.

**Start with a common trigger**  
Pick from one of the most commonly used triggers, then orchestrate any number of actions using the rich collection of connectors

When a message is received in a Service Bus queue	When a HTTP request is received	When a new tweet is posted
When an Event Grid resource event occurs	Recurrence	When a new email is received in Outlook.com

- In the dialog for When an HTTP request is received, select Use sample payload to generate schema.

When a HTTP request is received

HTTP POST URL  
URL will be generated after save

Request Body JSON Schema

Use sample payload to generate schema

Add new parameter

- Copy the following sample JSON into the textbox and select Done.

```
{
 "task": "<description>",
 "due": "<date>",
 "email": "<email-address>"
}
```

The schema is now generated for the request data you want. In practice, you can just capture the actual request data your application code generates and let Azure generate the JSON schema for you.

- At the top of the Logic Apps Designer, select Save.

You can now see the URL of your HTTP request trigger. Select the copy icon to copy it for later use.

The screenshot shows the Logic App designer interface. At the top, it says "When a HTTP request is received". Below that is an "HTTP POST URL" field containing "https://prod-112.westeurope.logic.azure.com:443/workflows/". To the right of the URL is a red-bordered "..." button. Underneath the URL is a "Request Body JSON Schema" section. It contains a JSON schema definition:

```
{ "type": "object", "properties": { "task": { "type": "string" }, "due": { "type": "string" } }}
```

Below the schema is a link "Use sample payload to generate schema". At the bottom of the configuration area is a "Add new parameter" dropdown.

This HTTP request definition is a trigger to anything you want to do in this logic app, be it Gmail or anything else. Later you will invoke this URL in your App Service app. For more information on the request trigger, see the [HTTP request/response reference](#).

6. At the bottom of the designer, click **New step**, type **Gmail** in the actions search box. Find and select **Send email (V2)**.

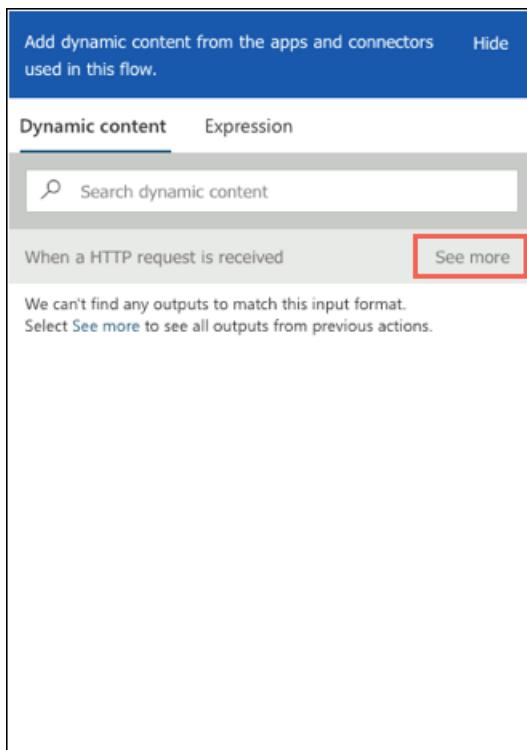
**TIP**

You can search for other types of integrations, such as SendGrid, MailChimp, Microsoft 365, and SalesForce. For more information, see [Logic Apps documentation](#).

7. In the **Gmail** dialog, select **Sign in** and sign in to the Gmail account you want to send the email from.

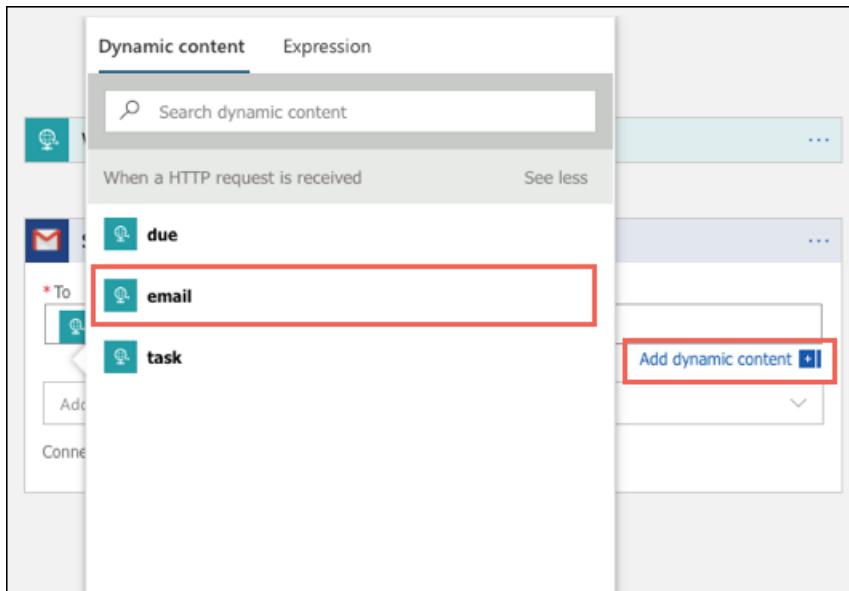


8. Once signed in, click in the **To** textbox, and the dynamic content dialog is automatically opened.
9. Next to the **When an HTTP request is received** action, select **See more**.



You should now see the three properties from your sample JSON data you used earlier. In this step, you use these properties from the HTTP request to construct an email.

10. Since you're selecting the value for the **To** field, choose **email**. If you want, toggle off the dynamic content dialog by clicking **Add dynamic content**.



11. In the **Add new parameter** dropdown, select **Subject** and **Body**.
12. Click in the **Subject** textbox, and in the same way, choose **task**. With the cursor still in the **Subject** box, type *created*.
13. Click in the **Body**, and in the same way, choose **due**. Move the cursor to the left of **due** and type *This work item is due on*.

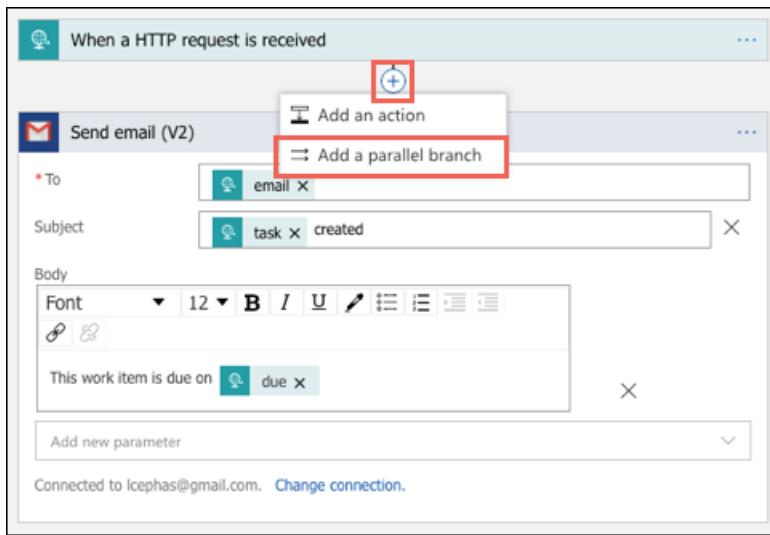
#### TIP

If you want to edit HTML content directly in the email body, select **Code view** at the top of the Logic Apps Designer window. Just make sure you preserve the dynamic content code (for example,

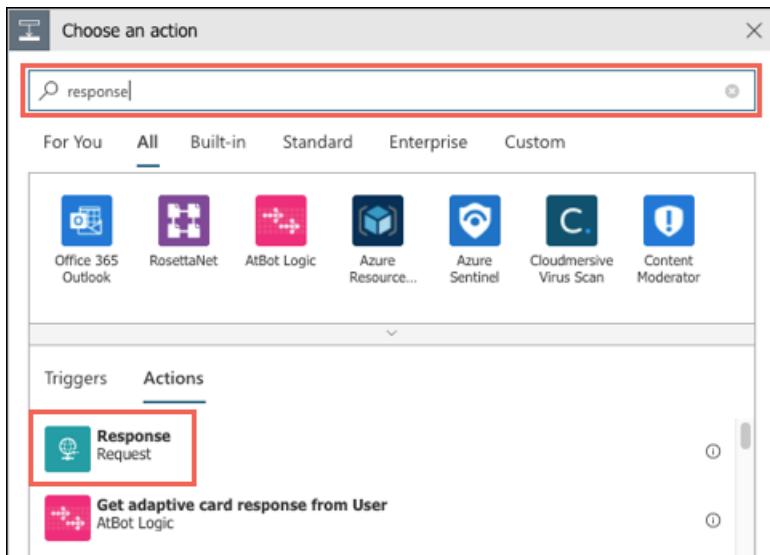
```
@{triggerBody()?['due']}
```

```
{
 "definition": {
 "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",
 "actions": {
 "Send_email_(V2)": {
 "inputs": {
 "body": {
 "Body": "<p>This work item is due on @{triggerBody()?['due']}</p>",
 "Subject": "@{triggerBody()?['task']} created",
 "To": "@{triggerBody()?['email']}"
 },
 "host": {
 "connection": {
 "name": "@parameters('$connections')['gmail']['connectionId']"
 }
 },
 "method": "post",
 "path": "/v2/Mail"
 },
 "runAfter": {},
 "type": "ApiConnection"
 }
 },
 "contentVersion": "1.0.0.0"
 }
}
```

14. Next, add an asynchronous HTTP response to the HTTP trigger. Between the HTTP trigger and the Gmail action, click the + sign and select **Add a parallel branch**.



15. In the search box, search for **response**, then select the **Response** action.



By default, the response action sends an HTTP 200. That's good enough for this tutorial. For more information, see the [HTTP request/response reference](#).

16. At the top of the Logic Apps Designer, select **Save** again.

## Add HTTP request code to app

Make sure you have copied the URL of the HTTP request trigger from earlier. Because it contains sensitive information, it's best practice that you don't put it into the code directly. With App Service, you can reference it as an environment variable instead, using app settings.

In the [Cloud Shell](#), create the app setting with the following command (replace `<app-name>`, `<resource-group-name>`, and `<logic-app-url>`):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings LOGIC_APP_URL=<your-logic-app-url>"
```

In your code, make a standard HTTP post to the URL using any HTTP client language that's available to your language framework, with the following configuration:

- The request body contains the same JSON format that you supplied to your logic app:

```
{
 "task": "<description>",
 "due": "<date>",
 "email": "<email-address>"
}
```

- The request contains the heading `Content-Type: application/json`.
- To optimize performance, send the request asynchronously if possible.

Click on the preferred language/framework tab below to see an example.

- [ASP.NET](#)
- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Ruby](#)

In ASP.NET, you can send the HTTP post with the [System.Net.Http.HttpClient](#) class. For example:

```
// requires using System.Net.Http;
var client = new HttpClient();
// requires using System.Text.Json;
var jsonData = JsonSerializer.Serialize(new
{
 email = "a-valid@emailaddress.com",
 due = "4/1/2020",
 task = "My new task!"
});

HttpResponseMessage result = await client.PostAsync(
 // requires using System.Configuration;
 ConfigurationManager.AppSettings["LOGIC_APP_URL"],
 new StringContent(jsonData, Encoding.UTF8, "application/json"));

var statusCode = result.StatusCode.ToString();
```

If you're testing this code on the sample app for [Tutorial: Build an ASP.NET app in Azure with SQL Database](#), you could use it to send an email confirmation in the [Create action](#), after the `Todo` item is added. To use the asynchronous code above, convert the Create action to asynchronous.

## More resources

[Tutorial: Host a RESTful API with CORS in Azure App Service](#)

[HTTP request/response reference for Logic Apps](#)

[Quickstart: Create your first workflow by using Azure Logic Apps - Azure portal](#)

- [Environment variables and app settings reference](#)

# Tutorial: Troubleshoot an App Service app with Azure Monitor

11/2/2021 • 6 minutes to read • [Edit Online](#)

This tutorial shows how to troubleshoot an [App Service](#) app using [Azure Monitor](#). The sample app includes code meant to exhaust memory and cause HTTP 500 errors, so you can diagnose and fix the problem using Azure Monitor. When you're finished, you'll have a sample app running on App Service on Linux integrated with [Azure Monitor](#).

[Azure Monitor](#) maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments.

In this tutorial, you learn how to:

- Configure a web app with Azure Monitor
- Send console logs to Log Analytics
- Use Log queries to identify and troubleshoot web app errors

You can follow the steps in this tutorial on macOS, Linux, Windows.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial, you'll need:

- [Azure subscription](#)
- [Azure CLI](#)
- [Git](#)

## Create Azure resources

First, you run several commands locally to setup a sample app to use with this tutorial. The commands create Azure resources, create a deployment user, and deploy the sample app to Azure. You'll be prompted for the password supplied as a part of the creation of the deployment user.

```
az group create --name myResourceGroup --location "South Central US"
az webapp deployment user set --user-name <username> --password <password>
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku B1 --is-linux
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.3" --deployment-local-git
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings
DEPLOYMENT_BRANCH='main'
git clone https://github.com/Azure-Samples/App-Service-Troubleshoot-Azure-Monitor
cd App-Service-Troubleshoot-Azure-Monitor
git branch -m main
git remote add azure <url-from-app-webapp-create>
git push azure main
```

## Configure Azure Monitor

## Create a Log Analytics Workspace

Now that you've deployed the sample app to Azure App Service, you'll configure monitoring capability to troubleshoot the app when problems arise. Azure Monitor stores log data in a Log Analytics workspace. A workspace is a container that includes data and configuration information.

In this step, you create a Log Analytics workspace to configure Azure Monitor with your app.

```
az monitor log-analytics workspace create --resource-group myResourceGroup --workspace-name
myMonitorWorkspace
```

### NOTE

For Azure Monitor Log Analytics, you pay for data ingestion and data retention.

## Create a diagnostic setting

Diagnostic settings can be used to collect metrics for certain Azure services into Azure Monitor Logs for analysis with other monitoring data using log queries. For this tutorial, you enable the web server and standard output/error logs. See [supported log types](#) for a complete list of log types and descriptions.

You run the following commands to create diagnostic settings for AppServiceConsoleLogs (standard output/error) and AppServiceHTTPLogs (web server logs). Replace <app-name> and <workspace-name> with your values.

### NOTE

The first two commands, `resourceID` and `workspaceID`, are variables to be used in the

```
az monitor diagnostic-settings create
```

 command. See [Create diagnostic settings using Azure CLI](#) for more information on this command.

```
resourceID=$(az webapp show -g myResourceGroup -n <app-name> --query id --output tsv)

workspaceID=$(az monitor log-analytics workspace show -g myResourceGroup --workspace-name <workspace-name> -
-query id --output tsv)

az monitor diagnostic-settings create --resource $resourceID \
--workspace $workspaceID \
-n myMonitorLogs \
--logs '[{"category": "AppServiceConsoleLogs", "enabled": true},
{"category": "AppServiceHTTPLogs", "enabled": true}]'
```

## Troubleshoot the app

Browse to <http://<app-name>.azurewebsites.net>.

The sample app, ImageConverter, converts included images from `JPG` to `PNG`. A bug has been deliberately placed in the code for this tutorial. If you select enough images, the the app produces a HTTP 500 error during image conversion. Imagine this scenario wasn't considered during the development phase. You'll use Azure Monitor to troubleshoot the error.

### Verify the app is works

To convert images, click `Tools` and select `Convert to PNG`.

View Images

Convert to PNG

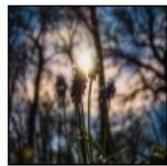
ImageConverter is a sample app for Azure App Service.

This app should be used along with this [tutorial](#).

Select the first two images and click `convert`. This will convert successfully.

### Select JPGs to convert to PNG

x



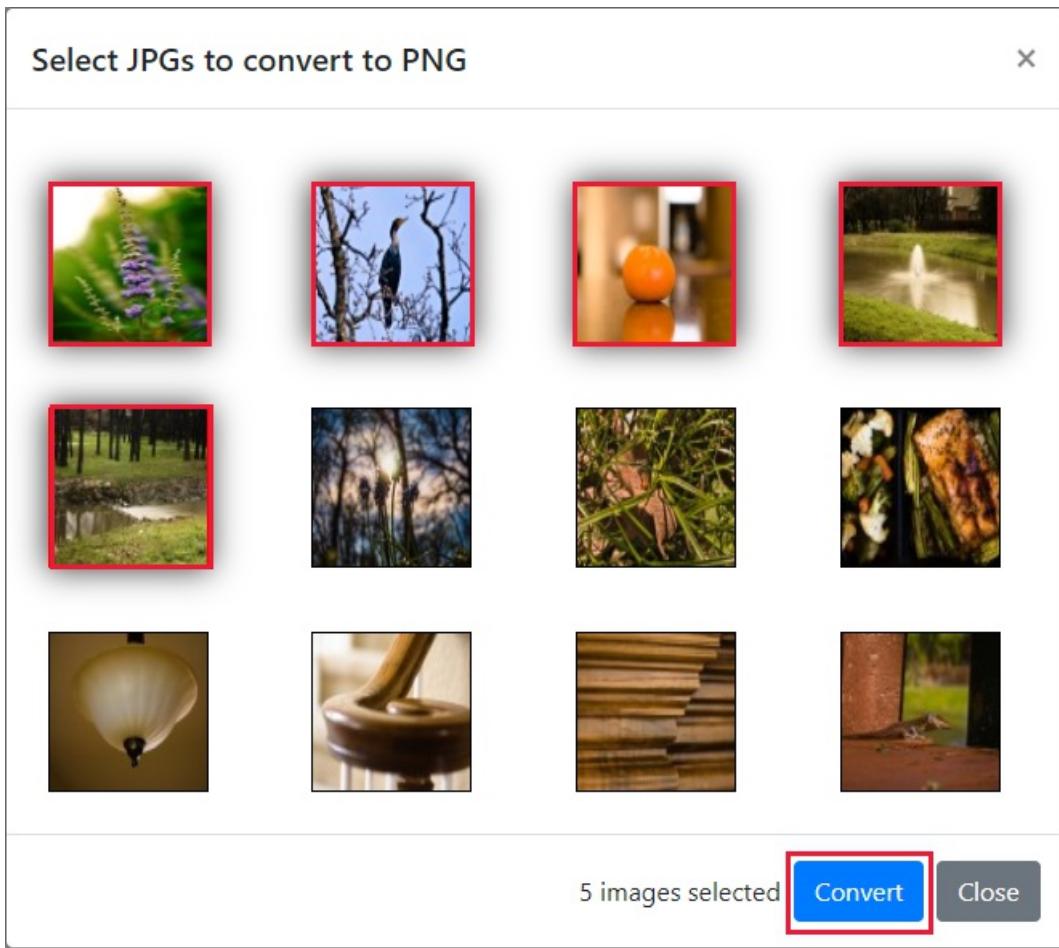
2 images selected

Convert

Close

### Break the app

Now that you've verified the app by converting two images successfully, we'll try to convert the first five images.



This action fails and produces a `HTTP 500` error that wasn't tested during development.



## Use log query to view Azure Monitor logs

Let's see what logs are available in the Log Analytics workspace.

Click this [Log Analytics workspace link](#) to access your workspace in the Azure portal.

In the Azure portal, select your Log Analytics workspace.

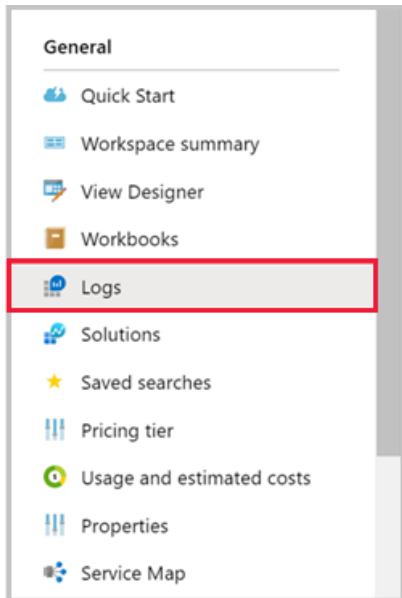
### Log queries

Log queries help you to fully apply the value of the data collected in Azure Monitor Logs. You use log queries to identify the logs in both AppServiceHTTPLogs and AppServiceConsoleLogs. See the [log query overview](#) for more information on log queries.

### View AppServiceHTTPLogs with log query

Now that we've accessed the app, let's view the data associated with HTTP requests, found in the [AppServiceHTTPLogs](#).

1. Click `Logs` from the left-hand navigation.



2. Search for `appservice` and double-click `AppServiceHTTPLogs`.

The screenshot shows the Azure Log Analytics workspace search interface. The search bar contains `appservice`. The results pane shows a table named `LogManagement` with several items listed. The item `AppServiceHTTPLogs` is highlighted with a yellow background and has a red border around it.

3. Click `Run`.

TimeGenerated [UTC]	Category	CsMethod	CsUriStem	Sport	Cip	UserAgent
2/27/2020, 10:08:59.000 PM	AppServiceHTTPLogs	GET	/process.php?images=3...	80	10.0.128.8	Mozilla/5.0 (Windows N
2/27/2020, 10:08:39.000 PM	AppServiceHTTPLogs	GET	/process.php?images=3...	80	10.0.128.7	Mozilla/5.0 (Windows N
2/27/2020, 10:07:44.000 PM	AppServiceHTTPLogs	GET	/	80	10.0.128.7	Mozilla/5.0 (Windows N
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/	80	10.0.128.8	Mozilla/5.0 (Windows N
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/starter-template.css	80	10.0.128.8	Mozilla/5.0 (Windows N
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/thumbs/img01.jpg	80	10.0.128.14	Mozilla/5.0 (Windows N
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/thumbs/img02.jpg	80	10.0.128.12	Mozilla/5.0 (Windows N
2/27/2020, 10:07:42.000 PM	AppServiceHTTPLogs	GET	/thumbs/img03.jpg	80	10.0.128.14	Mozilla/5.0 (Windows N

The `AppServiceHTTPLogs` query returns all requests in the past 24-hours. The column `scStatus` contains the HTTP status. To diagnose the `HTTP 500` errors, limit the `scStatus` to 500 and run the query, as shown below:

```
AppServiceHTTPLogs
| where ScStatus == 500
```

## View AppServiceConsoleLogs with log query

Now that you've confirmed the HTTP 500s, let's take a look at the standard output/errors from the app. These logs are found in `AppServiceConsoleLogs`.

- (1) Click `+` to create a new query.
- (2) Double-click the `AppServiceConsoleLogs` table and click `Run`.

Since converting five images results in server errors, you can see if the app is also writing errors by filtering `ResultDescription` for errors, as show below:

```
AppServiceConsoleLogs |
where ResultDescription contains "error"
```

In the `ResultDescription` column, you'll see the following error:

```
PHP Fatal error: Allowed memory size of 134217728 bytes exhausted
(tried to allocate 16384 bytes) in /home/site/wwwroot/process.php on line 20,
referer: http://<app-name>.azurewebsites.net/
```

## Join AppServiceHTTPLogs and AppServiceConsoleLogs

Now that you've identified both HTTP 500s and standard errors, you need to confirm if there's a correlation between these messages. Next, you join the tables together based on the time stamp, `TimeGenerated`.

## NOTE

A query has been prepared for you that does the following:

- Filters HTTPLogs for 500 errors
- Queries console logs
- Joins the tables on `TimeGenerated`

Run the following query:

```
let myHttp = AppServiceHTTPLogs | where ScStatus == 500 | project TimeGen=substring(TimeGenerated, 0, 19), CsUriStem, ScStatus;

let myConsole = AppServiceConsoleLogs | project TimeGen=substring(TimeGenerated, 0, 19), ResultDescription;

myHttp | join myConsole on TimeGen | project TimeGen, CsUriStem, ScStatus, ResultDescription;
```

In the `ResultDescription` column, you'll see the following error at the same time as web server errors:

```
PHP Fatal error: Allowed memory size of 134217728 bytes exhausted
(tried to allocate 16384 bytes) in /home/site/wwwroot/process.php on line 20,
referer: http://<app-name>.azurewebsites.net/
```

The message states memory has been exhausted on line 20 of `process.php`. You've now confirmed that the application produced an error during the HTTP 500 error. Let's take a look at the code to identify the problem.

## Identify the error

In the local directory, open the `process.php` and look at line 20.

```
imagepng($imgArray[$x], $filename);
```

The first argument, `$imgArray[$x]`, is a variable holding all JPGs (in-memory) needing conversion. However, `imagepng` only needs the image being converted and not all images. Pre-loading images is not necessary and may be causing the memory exhaustion, leading to HTTP 500s. Let's update the code to load images on-demand to see if it resolves the issue. Next, you will improve the code to address the memory problem.

## Fix the app

### Update locally and redeploy the code

You make the following changes to `process.php` to handle the memory exhaustion:

```
<?php

//Retrieve query parameters
$maxImages = $_GET['images'];
$imgNames = explode(",",$_GET['imgNames']);

//Load JPEGs into an array (in memory)
for ($x=0; $x<$maxImages; $x++){
 $filename = './images/converted_' . substr($imgNames[$x],0,-4) . '.png';
 imagepng(imagecreatefromjpeg("./images/" . $imgNames[$x]), $filename);
}
```

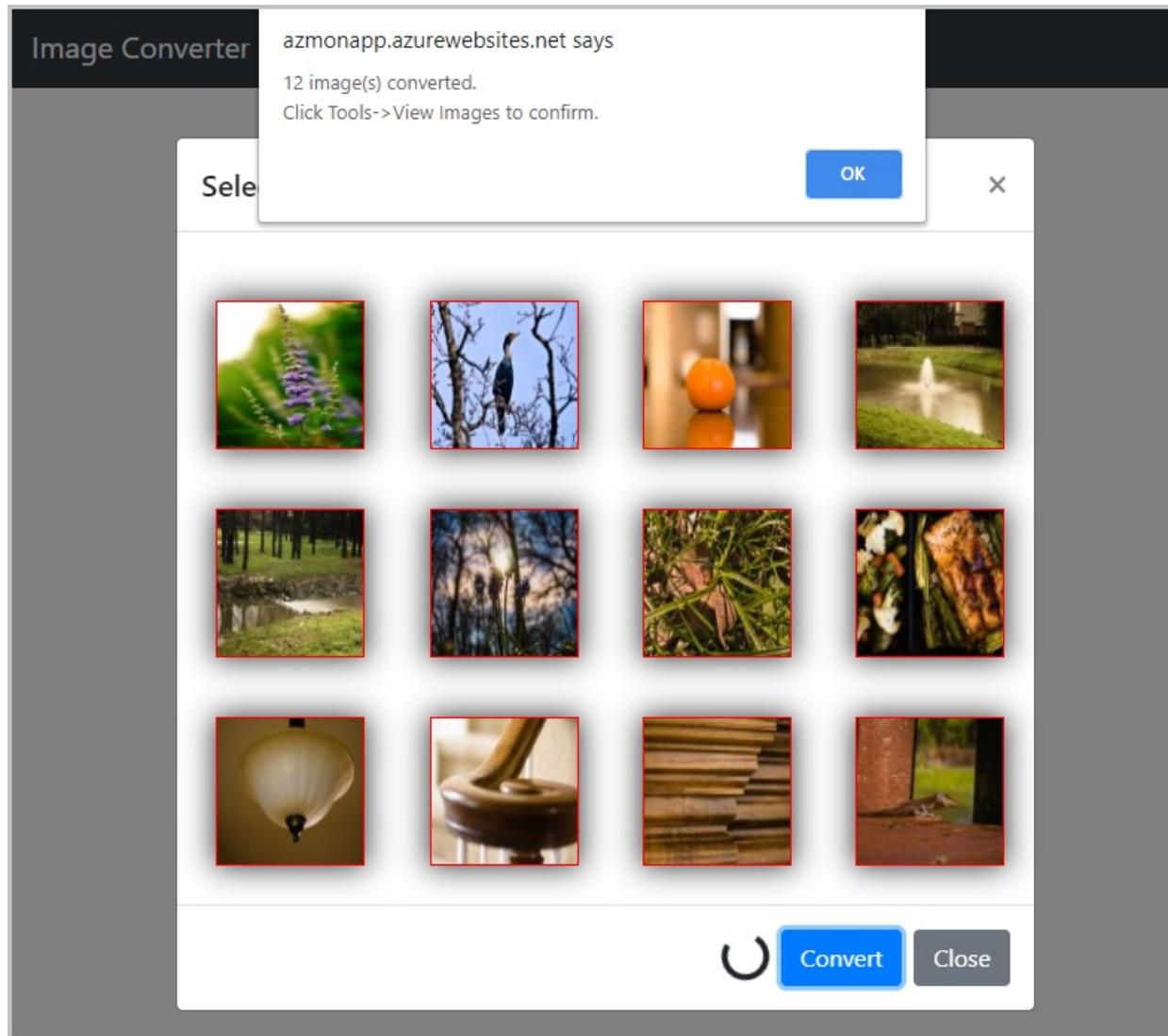
Commit your changes in Git, and then push the code changes to Azure.

```
git commit -am "Load images on-demand in process.php"
git push azure main
```

## Browse to the Azure app

Browse to <http://<app-name>.azurewebsites.net>.

Converting images should no longer produce the HTTP 500 errors.



## Clean up resources

In the preceding steps, you created Azure resources in a resource group. If you don't expect to need these resources in the future, delete the resource group by running the following command in the Cloud Shell:

```
az group delete --name myResourceGroup
```

This command may take a minute to run.

Delete the diagnostic setting with the following command:

```
az monitor diagnostic-settings delete --resource $resourceID -n myMonitorLogs
```

What you learned:

- Configured a web app with Azure Monitor
- Sent logs to Log Analytics
- Used log queries to identify and troubleshoot web app errors

## Next steps

- [Query logs with Azure Monitor](#)
- [Troubleshooting Azure App Service in Visual Studio](#)
- [Analyze app Logs in HDInsight](#)

# Tutorial: Use GitHub Actions to deploy to an App Service custom container and connect to a database

11/2/2021 • 4 minutes to read • [Edit Online](#)

This tutorial walks you through setting up a GitHub Actions workflow to deploy a containerized ASP.NET Core application with an [Azure SQL Database](#) backend. When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database. You'll first create Azure resources with an [ARM template](#) GitHub Actions workflow.

In this tutorial, you learn how to:

- Use a GitHub Actions workflow to add resources to Azure with a Azure Resource Manager template (ARM template)
- Use a GitHub Actions workflow to build a container with the latest web app changes

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial, you'll need:

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).
  - A GitHub repository to store your Resource Manager templates and your workflow files. To create one, see [Creating a new repository](#).

## Download the sample

[Fork the sample project](#) in the Azure Samples repo.

```
https://github.com/Azure-Samples/dotnetcore-containerized-sqldb-gactions/
```

## Create the resource group

Open the Azure Cloud Shell at <https://shell.azure.com>. You can alternately use the Azure CLI if you've installed it locally. (For more information on Cloud Shell, see the [Cloud Shell Overview](#).)

```
az group create --name {resource-group-name} --location {resource-group-location}
```

## Generate deployment credentials

You'll need to authenticate with a service principal for the resource deployment script to work. You can create a [service principal](#) with the `az ad sp create-for-rbac` command in the [Azure CLI](#). Run this command with [Azure Cloud Shell](#) in the Azure portal or by selecting the Try it button.

```
az ad sp create-for-rbac --name "{service-principal-name}" --sdk-auth --role contributor --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group-name}
```

In the example, replace the placeholders with your subscription ID, resource group name, and service principal name. The output is a JSON object with the role assignment credentials that provide access to your App Service app. Copy this JSON object for later. For help, go to [configure deployment credentials](#).

```
{
 "clientId": "<GUID>",
 "clientSecret": "<GUID>",
 "subscriptionId": "<GUID>",
 "tenantId": "<GUID>",
 (...)
}
```

#### IMPORTANT

It is always a good practice to grant minimum access. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

## Configure the GitHub secret for authentication

In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**.

To use [user-level credentials](#), paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret the name `AZURE_CREDENTIALS`.

## Add a SQL Server secret

Create a new secret in your repository for `SQL_SERVER_ADMIN_PASSWORD`. This secret can be any password that meets the Azure standards for password security. You won't be able to access this password again so save it separately.

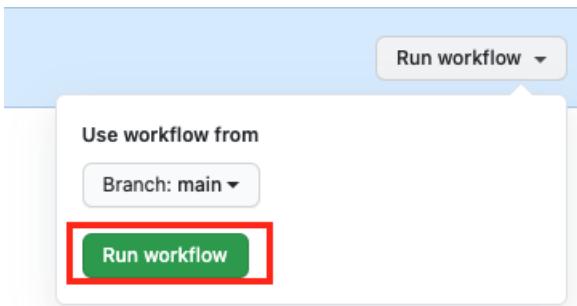
## Create Azure resources

The create Azure resources workflow runs an [ARM template](#) to deploy resources to Azure. The workflow:

- Checks out source code with the [Checkout action](#).
- Logs into Azure with the [Azure Login action](#) and gathers environment and Azure resource information.
- Deploys resources with the [Azure Resource Manager Deploy action](#).

To run the create Azure resources workflow:

1. Open the `azuredeploy.yaml` file in `.github/workflows` within your repository.
2. Update the value of `AZURE_RESOURCE_GROUP` to your resource group name.
3. Go to **Actions** and select **Run workflow**.



4. Verify that your action ran successfully by checking for a green checkmark on the **Actions** page.

### ✓ Create Azure Resources

## Add container registry and SQL secrets

1. In the Azure portal, open your newly created Azure Container Registry in your resource group.
2. Go to **Access keys** and copy the username and password values.
3. Create new GitHub secrets for `ACR_USERNAME` and `ACR_PASSWORD` password in your repository.
4. In the Azure portal, open your Azure SQL database. Open **Connection strings** and copy the value.
5. Create a new secret for `SQL_CONNECTION_STRING`. Replace `{your_password}` with your `SQL_SERVER_ADMIN_PASSWORD`.

## Build, push, and deploy your image

The build, push, and deploy workflow builds a container with the latest app changes, pushes the container to [Azure Container Registry](#) and, updates the web application staging slot to point to the latest container pushed. The workflow contains a build and deploy job:

- The build job checks out source code with the [Checkout action](#). The job then uses the [Docker login action](#) and a custom script to authenticate with Azure Container Registry, build a container image, and deploy it to Azure Container Registry.
- The deployment job logs into Azure with the [Azure Login action](#) and gathers environment and Azure resource information. The job then updates Web App Settings with the [Azure App Service Settings action](#) and deploys to an App Service staging slot with the [Azure Web Deploy action](#). Last, the job runs a custom script to update the SQL database and swaps staging slot to production.

To run the build, push, and deploy workflow:

1. Open your `build-deploy.yaml` file in `.github/workflows` within your repository.
2. Verify that the environment variables for `AZURE_RESOURCE_GROUP` and `WEB_APP_NAME` match the ones in `azuredeploy.yaml`.
3. Update the `ACR_LOGIN_SERVER` value for your Azure Container Registry login server.

## Next steps

[Learn about Azure and GitHub integration](#)

# Tutorial: Use GitHub Actions to deploy to App Service and connect to a database

11/2/2021 • 3 minutes to read • [Edit Online](#)

Learn how to set up a GitHub Actions workflow to deploy a ASP.NET Core application with an [Azure SQL Database](#) backend. When you're finished, you have an ASP.NET app running in Azure and connected to SQL Database. You'll first use an [ARM template](#) to create resources.

This tutorial does not use containers. If you want to deploy to a containerized ASP.NET Core application, see [Use GitHub Actions to deploy to App Service for Containers and connect to a database](#).

In this tutorial, you learn how to:

- Use a GitHub Actions workflow to add resources to Azure with a Azure Resource Manager template (ARM template)
- Use a GitHub Actions workflow to build an ASP.NET Core application

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

To complete this tutorial, you'll need:

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).
  - A GitHub repository to store your Resource Manager templates and your workflow files. To create one, see [Creating a new repository](#).

## Download the sample

[Fork the sample project](#) in the Azure Samples repo.

```
https://github.com/Azure-Samples/dotnetcore-sqldb-gactions
```

## Create the resource group

Open the Azure Cloud Shell at <https://shell.azure.com>. You can alternately use the Azure CLI if you've installed it locally. (For more information on Cloud Shell, see the [Cloud Shell Overview](#).)

```
az group create --name {resource-group-name} --location {resource-group-location}
```

## Generate deployment credentials

You'll need to authenticate with a service principal for the resource deployment script to work. You can create a [service principal](#) with the `az ad sp create-for-rbac` command in the [Azure CLI](#). Run this command with [Azure Cloud Shell](#) in the Azure portal or by selecting the Try it button.

```
az ad sp create-for-rbac --name "{service-principal-name}" --sdk-auth --role contributor --scopes /subscriptions/{subscription-id}
```

In the example, replace the placeholders with your subscription ID, resource group name, and service principal name. The output is a JSON object with the role assignment credentials that provide access to your App Service app. Copy this JSON object for later. For help, go to [configure deployment credentials](#).

```
{
 "clientId": "<GUID>",
 "clientSecret": "<GUID>",
 "subscriptionId": "<GUID>",
 "tenantId": "<GUID>",
 (...)
}
```

## Configure the GitHub secret for authentication

In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**.

To use [user-level credentials](#), paste the entire JSON output from the Azure CLI command into the secret's value field. Name the secret `AZURE_CREDENTIALS`.

## Add GitHub secrets for your build

1. Create [two new secrets](#) in your GitHub repository for `SQLADMIN_PASS` and `SQLADMIN_LOGIN`. Make sure you choose a complex password, otherwise the create step for the SQL database server will fail. You won't be able to access this password again so save it separately.
2. Create an `AZURE_SUBSCRIPTION_ID` secret for your Azure subscription ID. If you do not know your subscription ID, use this command in the Azure Shell to find it. Copy the value in the `SubscriptionId` column.

```
az account list -o table
```

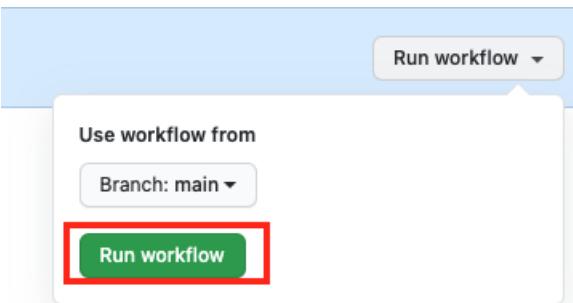
## Create Azure resources

The create Azure resources workflow runs an [ARM template](#) to deploy resources to Azure. The workflow:

- Checks out source code with the [Checkout action](#).
- Logs into Azure with the [Azure Login action](#) and gathers environment and Azure resource information.
- Deploys resources with the [Azure Resource Manager Deploy action](#).

To run the create Azure resources workflow:

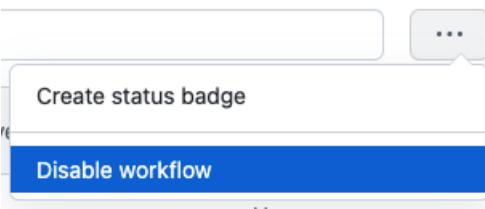
1. Open the `infraworkflow.yml` file in `.github/workflows` within your repository.
2. Update the value of `AZURE_RESOURCE_GROUP` to your resource group name.
3. Set the input for `region` in your ARM Deploy actions to your region.
  - a. Open `templates/azuredeploy.resourcegroup.parameters.json` and update the `rgLocation` property to your region.
4. Go to **Actions** and select **Run workflow**.



5. Verify that your action ran successfully by checking for a green checkmark on the **Actions** page.

### ✓ Create Azure Resources

6. After you've created your resources, go to **Actions**, select Create Azure Resources, disable the workflow.



## Create a publish profile secret

1. In the Azure portal, open your new staging App Service (Slot) created with the [Create Azure Resources](#) workflow.
2. Select **Get Publish Profile**.
3. Open the publish profile file in a text editor and copy its contents.
4. Create a new GitHub secret for `AZURE_WEBAPP_PUBLISH_PROFILE`.

## Build and deploy your app

To run the build and deploy workflow:

1. Open your `workflow.yaml` file in `.github/workflows` within your repository.
2. Verify that the environment variables for `AZURE_RESOURCE_GROUP`, `AZURE_WEBAPP_NAME`, `SQLSERVER_NAME`, and `DATABASE_NAME` match the ones in `infraworkflow.yml`.
3. Verify that your app deployed by visiting the URL in the Swap to production slot output. You should see a sample app, My TodoList App.

## Clean up resources

If you no longer need your sample project, delete your resource group in the Azure portal and delete your repository on GitHub.

## Next steps

[Learn about Azure and GitHub integration](#)

# CLI samples for Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure CLI.

SCRIPT	DESCRIPTION
<a href="#">Create app</a>	
<a href="#">Create an app and deploy files with FTP</a>	Creates an App Service app and deploys a file to it using FTP.
<a href="#">Create an app and deploy code from GitHub</a>	Creates an App Service app and deploys code from a public GitHub repository.
<a href="#">Create an app with continuous deployment from GitHub</a>	Creates an App Service app with continuous publishing from a GitHub repository you own.
<a href="#">Create an app and deploy code from a local Git repository</a>	Creates an App Service app and configures code push from a local Git repository.
<a href="#">Create an app and deploy code to a staging environment</a>	Creates an App Service app with a deployment slot for staging code changes.
<a href="#">Create an ASP.NET Core app in a Docker container</a>	Creates an App Service app on Linux and loads a Docker image from Docker Hub.
<a href="#">Create an app and expose it with a Private Endpoint</a>	Creates an App Service app and a Private Endpoint
<a href="#">Configure app</a>	
<a href="#">Map a custom domain to an app</a>	Creates an App Service app and maps a custom domain name to it.
<a href="#">Bind a custom TLS/SSL certificate to an app</a>	Creates an App Service app and binds the TLS/SSL certificate of a custom domain name to it.
<a href="#">Scale app</a>	
<a href="#">Scale an app manually</a>	Creates an App Service app and scales it across 2 instances.
<a href="#">Scale an app worldwide with a high-availability architecture</a>	Creates two App Service apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
<a href="#">Protect app</a>	
<a href="#">Integrate with Azure Application Gateway</a>	Creates an App Service app and integrates it with Application Gateway using service endpoint and access restrictions.
<a href="#">Connect app to resources</a>	

SCRIPT	DESCRIPTION
<a href="#">Connect an app to a SQL Database</a>	Creates an App Service app and a database in Azure SQL Database, then adds the database connection string to the app settings.
<a href="#">Connect an app to a storage account</a>	Creates an App Service app and a storage account, then adds the storage connection string to the app settings.
<a href="#">Connect an app to an Azure Cache for Redis</a>	Creates an App Service app and an Azure Cache for Redis, then adds the redis connection details to the app settings.)
<a href="#">Connect an app to Cosmos DB</a>	Creates an App Service app and a Cosmos DB, then adds the Cosmos DB connection details to the app settings.
<b>Backup and restore app</b>	
<a href="#">Backup an app</a>	Creates an App Service app and creates a one-time backup for it.
<a href="#">Create a scheduled backup for an app</a>	Creates an App Service app and creates a scheduled backup for it.
<a href="#">Restores an app from a backup</a>	Restores an App Service app from a backup.
<b>Monitor app</b>	
<a href="#">Monitor an app with web server logs</a>	Creates an App Service app, enables logging for it, and downloads the logs to your local machine.

# PowerShell samples for Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

The following table includes links to PowerShell scripts built using the Azure PowerShell.

SCRIPT	DESCRIPTION
<a href="#">Create app</a>	
<a href="#">Create an app with deployment from GitHub</a>	Creates an App Service app that pulls code from GitHub.
<a href="#">Create an app with continuous deployment from GitHub</a>	Creates an App Service app that continuously deploys code from GitHub.
<a href="#">Create an app and deploy code with FTP</a>	Creates an App Service app and upload files from a local directory using FTP.
<a href="#">Create an app and deploy code from a local Git repository</a>	Creates an App Service app and configures code push from a local Git repository.
<a href="#">Create an app and deploy code to a staging environment</a>	Creates an App Service app with a deployment slot for staging code changes.
<a href="#">Create an app and expose your app with a Private Endpoint</a>	Creates an App Service app with a Private Endpoint.
<a href="#">Configure app</a>	
<a href="#">Map a custom domain to an app</a>	Creates an App Service app and maps a custom domain name to it.
<a href="#">Bind a custom TLS/SSL certificate to an app</a>	Creates an App Service app and binds the TLS/SSL certificate of a custom domain name to it.
<a href="#">Scale app</a>	
<a href="#">Scale an app manually</a>	Creates an App Service app and scales it across 2 instances.
<a href="#">Scale an app worldwide with a high-availability architecture</a>	Creates two App Service apps in two different geographical regions and makes them available through a single endpoint using Azure Traffic Manager.
<a href="#">Connect app to resources</a>	
<a href="#">Connect an app to a SQL Database</a>	Creates an App Service app and a database in Azure SQL Database, then adds the database connection string to the app settings.
<a href="#">Connect an app to a storage account</a>	Creates an App Service app and a storage account, then adds the storage connection string to the app settings.
<a href="#">Back up and restore app</a>	

SCRIPT	DESCRIPTION
<a href="#">Back up an app</a>	Creates an App Service app and creates a one-time backup for it.
<a href="#">Create a scheduled backup for an app</a>	Creates an App Service app and creates a scheduled backup for it.
<a href="#">Delete a backup for an app</a>	Deletes an existing backup for an app.
<a href="#">Restore an app from backup</a>	Restores an app from a previously completed backup.
<a href="#">Restore a backup across subscriptions</a>	Restores a web app from a backup in another subscription.
<b>Monitor app</b>	
<a href="#">Monitor an app with web server logs</a>	Creates an App Service app, enables logging for it, and downloads the logs to your local machine.

# Azure Resource Manager templates for App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

The following table includes links to Azure Resource Manager templates for Azure App Service. For recommendations about avoiding common errors when you're creating app templates, see [Guidance on deploying apps with Azure Resource Manager templates](#).

To learn about the JSON syntax and properties for App Services resources, see [Microsoft.Web resource types](#).

DEPLOYING AN APP	DESCRIPTION
<a href="#">App Service plan and basic Linux app</a>	Deploys an App Service app that is configured for Linux.
<a href="#">App Service plan and basic Windows app</a>	Deploys an App Service app that is configured for Windows.
<a href="#">App linked to a GitHub repository</a>	Deploys an App Service app that pulls code from GitHub.
<a href="#">App with custom deployment slots</a>	Deploys an App Service app with custom deployment slots/environments.
<a href="#">App with Private Endpoint</a>	Deploys an App Service app with a Private Endpoint.
<b>Configuring an app</b>	<b>Description</b>
<a href="#">App certificate from Key Vault</a>	Deploys an App Service app certificate from an Azure Key Vault secret and uses it for TLS/SSL binding.
<a href="#">App with a custom domain and SSL</a>	Deploys an App Service app with a custom host name, and gets an app certificate from Key Vault for TLS/SSL binding.
<a href="#">App with Java 8 and Tomcat 8</a>	Deploys an App Service app with Java 8 and Tomcat 8 enabled. You can then run Java applications in Azure.
<a href="#">App with regional VNet integration</a>	Deploys an App Service app with regional VNet integration enabled.
<b>Protecting an app</b>	<b>Description</b>
<a href="#">App integrated with Azure Application Gateway</a>	Deploys an App Service app and an Application Gateway, and isolates the traffic using service endpoint and access restrictions.
<b>Linux app with connected resources</b>	<b>Description</b>
<a href="#">App on Linux with MySQL</a>	Deploys an App Service app on Linux with Azure Database for MySQL.
<a href="#">App on Linux with PostgreSQL</a>	Deploys an App Service app on Linux with Azure Database for PostgreSQL.
<b>App with connected resources</b>	<b>Description</b>

DEPLOYING AN APP	DESCRIPTION
<a href="#">App with MySQL</a>	Deploys an App Service app on Windows with Azure Database for MySQL.
<a href="#">App with PostgreSQL</a>	Deploys an App Service app on Windows with Azure Database for PostgreSQL.
<a href="#">App with a database in Azure SQL Database</a>	Deploys an App Service app and a database in Azure SQL Database at the Basic service level.
<a href="#">App with a Blob storage connection</a>	Deploys an App Service app with an Azure Blob storage connection string. You can then use Blob storage from the app.
<a href="#">App with an Azure Cache for Redis</a>	Deploys an App Service app with an Azure Cache for Redis.
<a href="#">App connected to a backend webapp</a>	Deploys two web apps (frontend and backend) securely connected together with VNet injection and Private Endpoint.
<b>App Service Environment</b>	<b>Description</b>
<a href="#">Create an App Service environment v2</a>	Creates an App Service environment v2 in your virtual network.
<a href="#">Create an App Service environment v2 with an ILB address</a>	Creates an App Service environment v2 in your virtual network with a private internal load balancer address.
<a href="#">Configure the default SSL certificate for an ILB App Service environment or an ILB App Service environment v2</a>	Configures the default TLS/SSL certificate for an ILB App Service environment or an ILB App Service environment v2.

# Terraform samples for Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

The following table includes links to terraform scripts.

SCRIPT	DESCRIPTION
<a href="#">Create app</a>	
<a href="#">Create two apps and connect securely with Private Endpoint and VNet integration</a>	Creates two App Service apps and connect apps together with Private Endpoint and VNet integration.
<a href="#">Provision App Service and use slot swap to deploy</a>	Provision App Service infrastructure with Azure deployment slots.

# Bicep files for App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

The following table includes links to Bicep files for Azure App Service. For quickstarts and further information about Bicep, see [Bicep documentation](#).

To learn about the Bicep syntax and properties for App Services resources, see [Microsoft.Web resource types](#).

DEPLOYING AN APP	DESCRIPTION
<a href="#">App Service plan and basic Linux app</a>	Deploys an App Service app that is configured for Linux.
<a href="#">App Service plan and basic Windows app</a>	Deploys an App Service app that is configured for Windows.
<a href="#">Website with container</a>	Deploys a website from a docker image configured for Linux.
<b>Configuring an app</b>	<b>Description</b>
<a href="#">App with conditional logging</a>	Deploys an App Service app with a conditional for logging enablement.
<a href="#">App with log analytics module</a>	Deploys an App Service app with log analytics.
<a href="#">App with regional VNet integration</a>	Deploys an App Service app with regional VNet integration enabled.
<b>App with connected resources</b>	<b>Description</b>
<a href="#">App with CosmosDB</a>	Deploys an App Service app on Linux with CosmosDB.
<a href="#">App with MySQL</a>	Deploys an App Service app on Windows with Azure Database for MySQL.
<a href="#">App with a database in Azure SQL Database</a>	Deploys an App Service app and a database in Azure SQL Database at the Basic service level.
<a href="#">App connected to a backend webapp</a>	Deploys two web apps (frontend and backend) securely connected together with VNet injection and Private Endpoint.
<a href="#">App with a database, managed identity, and monitoring</a>	Deploys an App Service App with a database, managed identity, and monitoring.
<b>App Service Environment</b>	<b>Description</b>
<a href="#">Create an App Service environment v2</a>	Creates an App Service environment v2 in your virtual network.

# Azure App Service plan overview

11/2/2021 • 7 minutes to read • [Edit Online](#)

In App Service (Web Apps, API Apps, or Mobile Apps), an app always runs in an *App Service plan*. In addition, [Azure Functions](#) also has the option of running in an *App Service plan*. An App Service plan defines a set of compute resources for a web app to run. 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 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:

- Operating System (Windows, Linux)
- 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. The pricing tiers available to your App Service plan depend on the operating system selected at creation time. There are a few categories of pricing tiers:

- **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 cannot 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.

## 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.

Each tier also provides a specific subset of App Service features. These features include custom domains and TLS/SSL certificates, autoscaling, deployment slots, backups, Traffic Manager integration, and more. The higher the tier, the more features are available. To find out which features are supported in each pricing tier, see [App Service plan details](#).

#### NOTE

The new **PremiumV3** pricing tier guarantees machines with faster processors (minimum 195 ACU per virtual CPU), SSD storage, and quadruple memory-to-core ratio compared to **Standard** tier. **PremiumV3** also supports higher scale via increased instance count while still providing all the advanced capabilities found in **Standard** tier. All features available in the existing **PremiumV2** tier are included in **PremiumV3**.

Similar to other dedicated tiers, three VM sizes are available for this tier:

- Small (2 CPU core, 8 GiB of memory)
- Medium (4 CPU cores, 16 GiB of memory)
- Large (8 CPU cores, 32 GiB of memory)

For **PremiumV3** pricing information, see [App Service Pricing](#).

To get started with the new **PremiumV3** pricing tier, see [Configure PremiumV3 tier for App Service](#).

## How does my app run and scale?

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

When you create an app in App Service, it is put into an App Service plan. When the app runs, it 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.

For information on scaling out an app, see [Scale instance count manually or automatically](#).

## How much does my App Service plan cost?

This section describes how App Service apps are billed. For detailed, region-specific pricing information, see [App Service Pricing](#).

Except for **Free** tier, an App Service plan carries a charge on the compute resources it uses.

- In the **Shared** tier, each app receives a quota of CPU minutes, so *each app* is charged for the CPU quota.
- In the dedicated compute tiers (**Basic**, **Standard**, **Premium**, **PremiumV2**, **PremiumV3**), the App Service plan defines the number of VM instances the apps are scaled to, so *each VM instance* in the App Service plan is charged. These VM instances are charged the same regardless how many apps are running on them. To avoid unexpected charges, see [Clean up an App Service plan](#).
- In the **Isolated** tier, the App Service Environment defines the number of isolated workers that run your apps, and *each worker* is charged. In addition, there's a flat Stamp Fee for the running the App Service Environment itself.

You don't get charged for using the App Service features that are available to you (configuring custom domains, TLS/SSL certificates, deployment slots, backups, etc.). The exceptions are:

- App Service Domains - you pay when you purchase one in Azure and when you renew it each year.
- App Service Certificates - you pay when you purchase one in Azure and when you renew it each year.
- IP-based TLS connections - There's an hourly charge for each IP-based TLS connection, but some **Standard**

tier or above gives you one IP-based TLS connection for free. SNI-based TLS connections are free.

#### NOTE

If you integrate App Service with another Azure service, you may need to consider charges from these other services. For example, if you use Azure Traffic Manager to scale your app geographically, Azure Traffic Manager also charges you based on your usage. To estimate your cross-services cost in Azure, see [Pricing calculator](#).

Want to optimize and save on your cloud spending?

Azure services cost money. Azure Cost Management helps you set budgets and configure alerts to keep spending under control. Analyze, manage, and optimize your Azure costs with Cost Management. To learn more, see the [quickstart on analyzing your costs](#).

## 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. You can choose a lower pricing tier at first and scale up later when you need more App Service features.

For example, you can start testing your web app in a **Free** App Service plan and pay nothing. When you want to add your [custom DNS name](#) to the web app, just scale your plan up to **Shared** tier. Later, when you want to [create a TLS binding](#), scale your plan up to **Basic** tier. When you want to have [staging environments](#), scale up to **Standard** tier. When you need more cores, memory, or storage, scale up to a bigger VM size in the same tier.

The same works in the reverse. When you feel you no longer need the capabilities or features of a higher tier, you can scale down to a lower tier, which saves you money.

For information on scaling up the App Service plan, see [Scale up an app in Azure](#).

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. For more information, see [Move an app to another App Service plan](#).

## Should I put an app in a new plan or an existing plan?

Since you pay for the computing resources your App Service plan allocates (see [How much does my App Service plan cost?](#)), you can potentially save money by putting multiple apps into one App Service plan. You can continue to add apps to an existing plan as long as the plan has enough resources to handle the load. However, keep in mind that apps in the same App Service plan all 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 potentially cause downtime for your new and existing apps.

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.

## Manage an App Service plan

[Manage an App Service plan](#)

# Plan and manage costs for Azure App Service

11/2/2021 • 9 minutes to read • [Edit Online](#)

This article describes how you plan for and manage costs for Azure App Service. First, you use the Azure pricing calculator to help plan for App Service costs before you add any resources for the service to estimate costs. Next, as you add Azure resources, review the estimated costs. After you've started using App Service resources, use [Cost Management](#) features to set budgets and monitor costs. You can also review forecasted costs and identify spending trends to identify areas where you might want to act. Costs for Azure App Service are only a portion of the monthly costs in your Azure bill. Although this article explains how to plan for and manage costs for App Service, you're billed for all Azure services and resources used in your Azure subscription, including the third-party services.

## Understand the full billing model for Azure App Service

Azure App Service runs on Azure infrastructure that accrues costs when you deploy new resources. It's important to understand that there could be other additional infrastructure costs that might accrue.

### How you're charged for Azure App Service

When you create or use App Service resources, you're charged for the following meters:

- You're charged an hourly rate based on the pricing tier of your App Service plan, prorated to the second.
- The charge is applied to each scaled-out instance in your plan, based on the amount of time that the VM instance is allocated.

Other cost resources for App Service are (see [App Service pricing](#) for details):

- [App Service domains](#) Your subscription is charged for the domain registration on a yearly basis, if you enable automatic renewal.
- [App Service certificates](#) One-time charge at the time of purchase. If you have multiple subdomains to secure, you can reduce cost by purchasing one wildcard certificate instead of multiple standard certificates.
- [IP-based certificate bindings](#) The binding is configured on a certificate at the app level. Costs are accrued for each binding. For **Standard** tier and above, the first IP-based binding is not charged.

At the end of your billing cycle, the charges for each VM instance. Your bill or invoice shows a section for all App Service costs. There's a separate line item for each meter.

### Other costs that might accrue with Azure App Service

Depending on which feature you use in App Service, the following cost-accruing resources may be created:

- **Isolated tier** A [Virtual Network](#) is required for an App Service environment and is charged separately.
- **Backup** A [Storage account](#) is required to make backups and is charged separately.
- **Diagnostic logs** You can select [Storage account](#) as the logging option, or integrate with [Azure Log Analytics](#). These services are charged separately.
- **App Service certificates** Certificates you purchase in Azure must be maintained in [Azure Key Vault](#), which is charged separately.

### Costs that might accrue after resource deletion

When you delete all apps in an App Service plan, the plan continues to accrue charges based on its configured pricing tier and number of instances. To avoid unwanted charges, delete the plan or scale it down to **Free** tier.

After you delete Azure App Service resources, resources from related Azure services might continue to exist.

They continue to accrue costs until you delete them. For example:

- The Virtual Network that you created for an **Isolated** tier App Service plan
- Storage accounts you created to store backups or diagnostic logs
- Key Vault you created to store App Service certificates
- Log Analytic namespaces you created to ship diagnostic logs
- [Instance or stamp reservations](#) for App Service that haven't expired yet

## Using Azure Prepayment with Azure App Service

You can pay for Azure App Service charges with your Azure Prepayment credit. However, you can't use Azure Prepayment credit to pay for charges for third-party products and services, including those from the Azure Marketplace.

## Estimate costs

An easy way to estimate and optimize your App Service cost beforehand is by using the [Azure pricing calculator](#).

To use the pricing calculator, click **App Service** in the **Products** tab. Then, scroll down to work with the calculator. The following screenshot is an example and doesn't reflect current pricing.

The screenshot shows the Azure Pricing Calculator interface for estimating App Service costs. At the top, it displays "Your Estimate" and the selected product as "App Service". Below this, it shows the configuration: Region (West US), Operating System (Windows), and Tier (Premium V3). The instance selected is "P1V3: 2 Cores(s), 8 GB RAM, 250 GB Storage, \$0.335". The estimated cost is shown as "Upfront: \$0.00" and "Monthly: \$244.55".

**Premium V3**

INSTANCES: P1V3: 2 Cores(s), 8 GB RAM, 250 GB Storage, \$0.335

1 Instances    730 Hours    = \$244.55

**Savings Options**

Save up to 42% on pay as you go prices with 1 year or 3 year reserved instances

Pay as you go  
 1 year reserved (~25% discount)  
 3 year reserved (~40% discount)

\$244.55  
Average per month  
(\$0.00 charged upfront)

= \$244.55  
Average per month  
(\$0.00 charged upfront)

**SSL Connections**

	Upfront cost	Monthly cost
	\$0.00	\$244.55

**Support**

SUPPORT: Included    = \$0.00

**Programs and Offers**

LICENSING PROGRAM: Microsoft Online Services Agreement    = \$0.00

SHOW DEV/TEST PRICING

**Estimated upfront cost**    \$0.00  
**Estimated monthly cost**    \$244.55

## Review estimated costs in the Azure portal

When you create an App Service app or an App Service plan, you can see the estimated costs.

To create an app and view the estimated price:

1. On the create page, scroll down to **App Service plan**, and click **Create new**.
2. Specify a name and click **OK**.
3. Next to **Sku and size**, click **Change size**.
4. Review the estimated price shown in the summary. The following screenshot is an example and doesn't reflect current pricing.

**Spec Picker**

**Recommended pricing tiers**

<b>P1V2</b> 210 total ACU 3.5 GB memory Dv2-Series compute equivalent 81.03 USD/Month (Estimated)	<b>P2V2</b> 420 total ACU 7 GB memory Dv2-Series compute equivalent 161.33 USD/Month (Estimated)	<b>P3V2</b> 840 total ACU 14 GB memory Dv2-Series compute equivalent 322.66 USD/Month (Estimated)
<b>P1V3</b> 195 minimum ACU/vCPU 8 GB memory 2 vCPU 93.44 USD/Month (Estimated)	<b>P2V3</b> 195 minimum ACU/vCPU 16 GB memory 4 vCPU 186.88 USD/Month (Estimated)	<b>P3V3</b> 195 minimum ACU/vCPU 32 GB memory 8 vCPU 373.76 USD/Month (Estimated)

[See additional options](#)

**Included features**  
Every app hosted on this App Service plan will have access to these features:

- Custom domains / SSL**  
Configure and purchase custom domains with SNI and IP SSL bindings
- Auto scale**  
Up to 10 instances. Subject to availability.

**Included hardware**  
Every instance of your App Service plan will include the following hardware configuration:

- Azure Compute Units (ACU)**  
Dedicated compute resources used to run applications deployed in the App Service Plan. [Learn more](#)
- Memory**  
Memory per instance available to run applications deployed and running in the App Service plan.

If your Azure subscription has a spending limit, Azure prevents you from spending over your credit amount. As you create and use Azure resources, your credits are used. When you reach your credit limit, the resources that you deployed are disabled for the rest of that billing period. You can't change your credit limit, but you can remove it. For more information about spending limits, see [Azure spending limit](#).

## Optimize costs

At a basic level, App Service apps are charged by the App Service plan that hosts them. The costs associated with your App Service deployment depend on a few main factors:

- **Pricing tier** Otherwise known as the SKU of the App Service plan. Higher tiers provide more CPU cores, memory, storage, or features, or combinations of them.
- **Instance count** dedicated tiers (Basic and above) can be scaled out, and each scaled out instance accrues costs.
- **Stamp fee** In the Isolated tier, a flat fee is accrued on your App Service environment, regardless of how many apps or worker instances are hosted.

An App Service plan can host more than one app. Depending on your deployment, you could save costs hosting more apps on one App Service plans (i.e. hosting your apps on fewer App Service plans).

For details, see [App Service plan overview](#)

## Non-production workloads

To test App Service or your solution while accruing low or minimal cost, you can begin by using the two entry-level pricing tiers, **Free** and **Shared**, which are hosted on shared instances. To test your app on dedicated instances with better performance, you can upgrade to **Basic** tier, which supports both Windows and Linux apps.

#### **NOTE**

**Azure Dev/Test Pricing** To test pre-production workloads that require higher tiers (all tiers except for **Isolated**), Visual Studio subscribers can also take advantage of the [Azure Dev/Test Pricing](#), which offers significant discounts.

Both the **Free** and **Shared** tier, as well as the Azure Dev/Test Pricing discounts, don't carry a financially backed SLA.

## **Production workloads**

Production workloads come with the recommendation of the dedicated **Standard** pricing tier or above. While the price goes up for higher tiers, it also gives you more memory and storage and higher-performing hardware, giving you higher app density per compute instance. That translates to lower instance count for the same number of apps, and therefore lower cost. In fact, **Premium V3** (the highest non-**Isolated** tier) is the most cost effective way to serve your app at scale. To add to the savings, you can get deep discounts on [Premium V3 reservations](#).

#### **NOTE**

**Premium V3** supports both Windows containers and Linux containers.

Once you choose the pricing tier you want, you should minimize the idle instances. In a scale-out deployment, you can waste money on underutilized compute instances. You should [configure autoscaling](#), available in **Standard** tier and above. By creating scale-out schedules, as well as metric-based scale-out rules, you only pay for the instances you really need at any given time.

## **Azure Reservations**

If you plan to utilize a known minimum number of compute instances for one year or more, you should take advantage of **Premium V3** tier and drive down the instance cost drastically by reserving those instances in 1-year or 3-year increments. The monthly cost savings can be as much as 55% per instance. Two types of reservations are possible:

- **Windows (or platform agnostic)** Can apply to Windows or Linux instances in your subscription.
- **Linux specific** Applies only to Linux instances in your subscription.

The reserved instance pricing applies to the applicable instances in your subscription, up to the number of instances that you reserve. The reserved instances are a billing matter and are not tied to specific compute instances. If you run fewer instances than you reserve at any point during the reservation period, you still pay for the reserved instances. If you run more instances than you reserve at any point during the reservation period, you pay the normal accrued cost for the additional instances.

The **Isolated** tier (App Service environment) also supports 1-year and 3-year reservations at reduced pricing. For more information, see [How reservation discounts apply to Azure App Service](#).

## **Monitor costs**

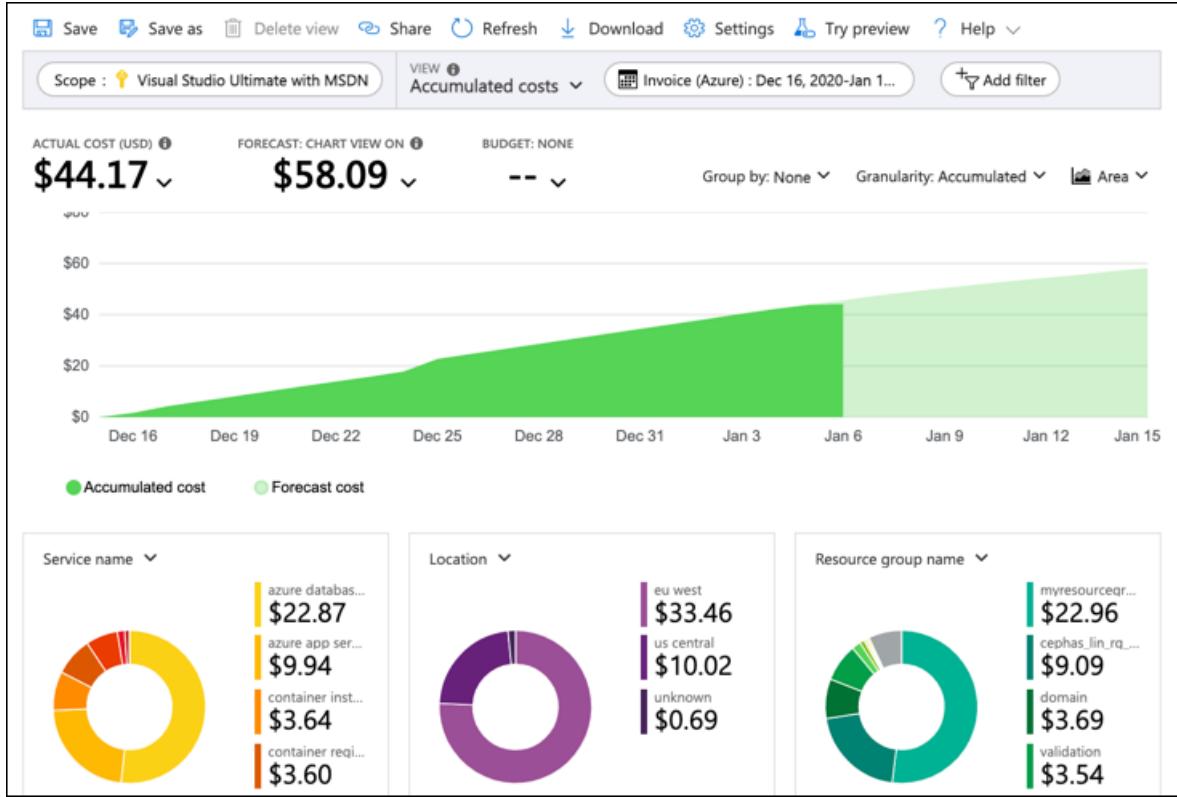
As you use Azure resources with App Service, you incur costs. Azure resource usage unit costs vary by time intervals (seconds, minutes, hours, and days). As soon as App Service use starts, costs are incurred and you can see the costs in [cost analysis](#).

When you use cost analysis, you view App Service costs in graphs and tables for different time intervals. Some examples are by day, current and prior month, and year. You also view costs against budgets and forecasted costs. Switching to longer views over time can help you identify spending trends. And you see where overspending might have occurred. If you've created budgets, you can also easily see where they're exceeded.

To view App Service costs in cost analysis:

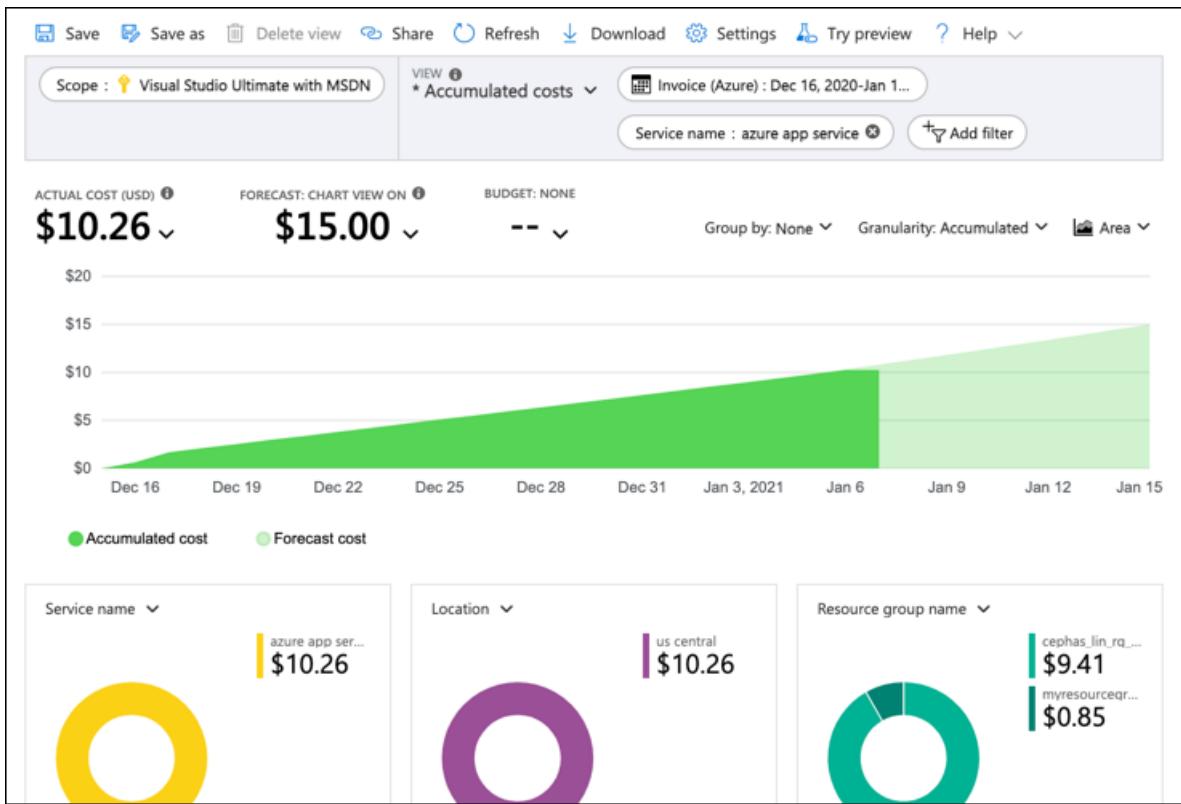
1. Sign in to the Azure portal.
2. Open the scope in the Azure portal and select **Cost analysis** in the menu. For example, go to **Subscriptions**, select a subscription from the list, and then select **Cost analysis** in the menu. Select **Scope** to switch to a different scope in cost analysis.
3. By default, cost for services are shown in the first donut chart. Select the area in the chart labeled App Service.

Actual monthly costs are shown when you initially open cost analysis. Here's an example showing all monthly usage costs.



To narrow costs for a single service, like App Service, select **Add filter** and then select **Service name**. Then, select **App Service**.

Here's an example showing costs for just App Service.



In the preceding example, you see the current cost for the service. Costs by Azure regions (locations) and App Service costs by resource group are also shown. From here, you can explore costs on your own.

## Create budgets

You can create [budgets](#) to manage costs and create [alerts](#) that automatically notify stakeholders of spending anomalies and overspending risks. Alerts are based on spending compared to budget and cost thresholds. Budgets and alerts are created for Azure subscriptions and resource groups, so they're useful as part of an overall cost monitoring strategy.

Budgets can be created with filters for specific resources or services in Azure if you want more granularity present in your monitoring. Filters help ensure that you don't accidentally create new resources that cost you extra money. For more information about the filter options available when you create a budget, see [Group and filter options](#).

## Export cost data

You can also [export your cost data](#) to a storage account. This is helpful when you need or others to do more data analysis for costs. For example, a finance team can analyze the data using Excel or Power BI. You can export your costs on a daily, weekly, or monthly schedule and set a custom date range. Exporting cost data is the recommended way to retrieve cost datasets.

## Next steps

- Learn more on how pricing works with Azure Storage. See [App Service pricing](#).
- Learn [how to optimize your cloud investment with Azure Cost Management](#).
- Learn more about managing costs with [cost analysis](#).
- Learn about how to [prevent unexpected costs](#).
- Take the [Cost Management](#) guided learning course.

# Operating system functionality on Azure App Service

11/2/2021 • 10 minutes to read • [Edit Online](#)

This article describes the common baseline operating system functionality that is available to all Windows apps running on [Azure App Service](#). This functionality includes file, network, and registry access, and diagnostics logs and events.

## NOTE

[Linux apps](#) in App Service run in their own containers. No access to the host operating system is allowed, you do have root access to the container. Likewise, for [apps running in Windows containers](#), you have administrative access to the container but no access to the host operating system.

## App Service plan tiers

App Service runs customer apps in a multi-tenant hosting environment. Apps deployed in the **Free** and **Shared** tiers run in worker processes on shared virtual machines, while apps deployed in the **Standard** and **Premium** tiers run on virtual machine(s) dedicated specifically for the apps associated with a single customer.

## 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.

Because App Service supports a seamless scaling experience between different tiers, the security configuration enforced for App Service apps remains the same. This ensures that apps don't suddenly behave differently, failing in unexpected ways, when App Service plan switches from one tier to another.

## Development frameworks

App Service pricing tiers control the amount of compute resources (CPU, disk storage, memory, and network egress) available to apps. However, the breadth of framework functionality available to apps remains the same regardless of the scaling tiers.

App Service supports a variety of development frameworks, including ASP.NET, classic ASP, Node.js, PHP, and Python - all of which run as extensions within IIS. In order to simplify and normalize security configuration, App Service apps typically run the various development frameworks with their default settings. One approach to configuring apps could have been to customize the API surface area and functionality for each individual development framework. App Service instead takes a more generic approach by enabling a common baseline of operating system functionality regardless of an app's development framework.

The following sections summarize the general kinds of operating system functionality available to App Service apps.

# File access

Various drives exist within App Service, including local drives and network drives.

## Local drives

At its core, App Service is a service running on top of the Azure PaaS (platform as a service) infrastructure. As a result, the local drives that are "attached" to a virtual machine are the same drive types available to any worker role running in Azure. This includes:

- An operating system drive (`%SystemDrive%`), whose size varies depending on the size of the VM.
- A resource drive (`%ResourceDrive%`) used by App Service internally.

It is important to monitor your disk utilization as your application grows. If the disk quota is reached, it can have adverse effects to your application. For example:

- The app may throw an error indicating not enough space on the disk.
- You may see disk errors when browsing to the Kudu console.
- Deployment from Azure DevOps or Visual Studio may fail with  
`ERROR_NOT_ENOUGH_DISK_SPACE: Web deployment task failed. (Web Deploy detected insufficient space on disk)`.
- Your app may suffer slow performance.

## Network drives (UNC shares)

One of the unique aspects of App Service that makes app deployment and maintenance straightforward is that all content shares are stored on a set of UNC shares. This model maps well to the common pattern of content storage used by on-premises web hosting environments that have multiple load-balanced servers.

Within App Service, there is a number of UNC shares created in each data center. A percentage of the user content for all customers in each data center is allocated to each UNC share. Each customer's subscription has a reserved directory structure on a specific UNC share within a data center. A customer may have multiple apps created within a specific data center, so all of the directories belonging to a single customer subscription are created on the same UNC share.

Due to how Azure services work, the specific virtual machine responsible for hosting a UNC share will change over time. It is guaranteed that UNC shares will be mounted by different virtual machines as they are brought up and down during the normal course of Azure operations. For this reason, apps should never make hard-coded assumptions that the machine information in a UNC file path will remain stable over time. Instead, they should use the convenient *faux* absolute path `%HOME%\site` that App Service provides. This faux absolute path provides a portable, app-and-user-agnostic method for referring to one's own app. By using `%HOME%\site`, one can transfer shared files from app to app without having to configure a new absolute path for each transfer.

## Types of file access granted to an app

The `%HOME%` directory in an app maps to a content share in Azure Storage dedicated for that app, and its size is defined by your [pricing tier](#). It may include directories such as those for content, error and diagnostic logs, and earlier versions of the app created by source control. These directories are available to the app's application code at runtime for read and write access. Because the files are not stored locally, they are persistent across app restarts.

On the system drive, App Service reserves `%SystemDrive%\local` for app-specific temporary local storage. Changes to files in this directory are *not* persistent across app restarts. Although an app has full read/write access to its own temporary local storage, that storage really isn't intended to be used directly by the application code. Rather, the intent is to provide temporary file storage for IIS and web application frameworks. App Service also limits the amount of storage in `%SystemDrive%\local` for each app to prevent individual apps from consuming excessive amounts of local file storage. For **Free**, **Shared**, and **Consumption** (Azure Functions)

tiers, the limit is 500 MB. See the following table for other tiers:

SKU FAMILY	B1/S1/ETC.	B2/S2/ETC.	B3/S3/ETC.
Basic, Standard, Premium	11 GB	15 GB	58 GB
PremiumV2, PremiumV3, Isolated	21 GB	61 GB	140 GB

Two examples of how App Service uses temporary local storage are the directory for temporary ASP.NET files and the directory for IIS compressed files. The ASP.NET compilation system uses the

`%SystemDrive%\local\Temporary ASP.NET Files` directory as a temporary compilation cache location. IIS uses the `%SystemDrive%\local\IIS Temporary Compressed Files` directory to store compressed response output. Both of these types of file usage (as well as others) are remapped in App Service to per-app temporary local storage. This remapping ensures that functionality continues as expected.

Each app in App Service runs as a random unique low-privileged worker process identity called the "application pool identity", described further here: <https://www.iis.net/learn/manage/configuring-security/application-pool-identities>. Application code uses this identity for basic read-only access to the operating system drive. This means application code can list common directory structures and read common files on operating system drive. Although this might appear to be a somewhat broad level of access, the same directories and files are accessible when you provision a worker role in an Azure hosted service and read the drive contents.

### File access across multiple instances

The content share (`%HOME%`) directory contains an app's content, and application code can write to it. If an app runs on multiple instances, the `%HOME%` directory is shared among all instances so that all instances see the same directory. So, for example, if an app saves uploaded files to the `%HOME%` directory, those files are immediately available to all instances.

The temporary local storage (`%SystemDrive%\local`) directory is not shared between instances, neither is it shared between the app and its [Kudu app](#).

## Network access

Application code can use TCP/IP and UDP-based protocols to make outbound network connections to Internet accessible endpoints that expose external services. Apps can use these same protocols to connect to services within Azure, for example, by establishing HTTPS connections to SQL Database.

There is also a limited capability for apps to establish one local loopback connection, and have an app listen on that local loopback socket. This feature exists primarily to enable apps that listen on local loopback sockets as part of their functionality. Each app sees a "private" loopback connection. App "A" cannot listen to a local loopback socket established by app "B".

Named pipes are also supported as an inter-process communication (IPC) mechanism between different processes that collectively run an app. For example, the IIS FastCGI module relies on named pipes to coordinate the individual processes that run PHP pages.

## Code execution, processes, and memory

As noted earlier, apps run inside of low-privileged worker processes using a random application pool identity. Application code has access to the memory space associated with the worker process, as well as any child processes that may be spawned by CGI processes or other applications. However, one app cannot access the memory or data of another app even if it is on the same virtual machine.

Apps can run scripts or pages written with supported web development frameworks. App Service doesn't configure any web framework settings to more restricted modes. For example, ASP.NET apps running on App Service run in "full" trust as opposed to a more restricted trust mode. Web frameworks, including both classic ASP and ASP.NET, can call in-process COM components (but not out of process COM components) like ADO (ActiveX Data Objects) that are registered by default on the Windows operating system.

Apps can spawn and run arbitrary code. It is allowable for an app to do things like spawn a command shell or run a PowerShell script. However, even though arbitrary code and processes can be spawned from an app, executable programs and scripts are still restricted to the privileges granted to the parent application pool. For example, an app can spawn an executable that makes an outbound HTTP call, but that same executable cannot attempt to unbind the IP address of a virtual machine from its NIC. Making an outbound network call is allowed to low-privileged code, but attempting to reconfigure network settings on a virtual machine requires administrative privileges.

## Diagnostics logs and events

Log information is another set of data that some apps attempt to access. The types of log information available to code running in App Service includes diagnostic and log information generated by an app that is also easily accessible to the app.

For example, W3C HTTP logs generated by an active app are available either on a log directory in the network share location created for the app, or available in blob storage if a customer has set up W3C logging to storage. The latter option enables large quantities of logs to be gathered without the risk of exceeding the file storage limits associated with a network share.

In a similar vein, real-time diagnostics information from .NET apps can also be logged using the .NET tracing and diagnostics infrastructure, with options to write the trace information to either the app's network share, or alternatively to a blob storage location.

Areas of diagnostics logging and tracing that aren't available to apps are Windows ETW events and common Windows event logs (for example, System, Application, and Security event logs). Since ETW trace information can potentially be viewable machine-wide (with the right ACLs), read and write access to ETW events are blocked. Developers might notice that API calls to read and write ETW events and common Windows event logs appear to work, but that is because App Service is "faking" the calls so that they appear to succeed. In reality, the application code has no access to this event data.

## Registry access

Apps have read-only access to much (though not all) of the registry of the virtual machine they are running on. In practice, this means registry keys that allow read-only access to the local Users group are accessible by apps. One area of the registry that is currently not supported for either read or write access is the HKEY\_CURRENT\_USER hive.

Write-access to the registry is blocked, including access to any per-user registry keys. From the app's perspective, write access to the registry should never be relied upon in the Azure environment since apps can (and do) get migrated across different virtual machines. The only persistent writeable storage that can be depended on by an app is the per-app content directory structure stored on the App Service UNC shares.

## Remote desktop access

App Service doesn't provide remote desktop access to the VM instances.

## More information

[Azure App Service sandbox](#) - The most up-to-date information about the execution environment of App Service.  
This page is maintained directly by the App Service development team.

# Deployment Best Practices

11/2/2021 • 7 minutes to read • [Edit Online](#)

Every development team has unique requirements that can make implementing an efficient deployment pipeline difficult on any cloud service. This article introduces the three main components of deploying to App Service: deployment sources, build pipelines, and deployment mechanisms. This article also covers some best practices and tips for specific language stacks.

## Deployment Components

### Deployment Source

A deployment source is the location of your application code. For production apps, the deployment source is usually a repository hosted by version control software such as [GitHub](#), [BitBucket](#), or [Azure Repos](#). For development and test scenarios, the deployment source may be a [project on your local machine](#). App Service also supports [OneDrive and Dropbox folders](#) as deployment sources. While cloud folders can make it easy to get started with App Service, it is not typically recommended to use this source for enterprise-level production applications.

### Build Pipeline

Once you decide on a deployment source, your next step is to choose a build pipeline. A build pipeline reads your source code from the deployment source and executes a series of steps (such as compiling code, minifying HTML and JavaScript, running tests, and packaging components) to get the application in a runnable state. The specific commands executed by the build pipeline depend on your language stack. These operations can be executed on a build server such as Azure Pipelines, or executed locally.

### Deployment Mechanism

The deployment mechanism is the action used to put your built application into the `/home/site/wwwroot` directory of your web app. The `/wwwroot` directory is a mounted storage location shared by all instances of your web app. When the deployment mechanism puts your application in this directory, your instances receive a notification to sync the new files. App Service supports the following deployment mechanisms:

- Kudu endpoints: [Kudu](#) is the open-source developer productivity tool that runs as a separate process in Windows App Service, and as a second container in Linux App Service. Kudu handles continuous deployments and provides HTTP endpoints for deployment, such as zipdeploy.
- FTP and WebDeploy: Using your [site or user credentials](#), you can upload files via [FTP](#) or WebDeploy. These mechanisms do not go through Kudu.

Deployment tools such as Azure Pipelines, Jenkins, and editor plugins use one of these deployment mechanisms.

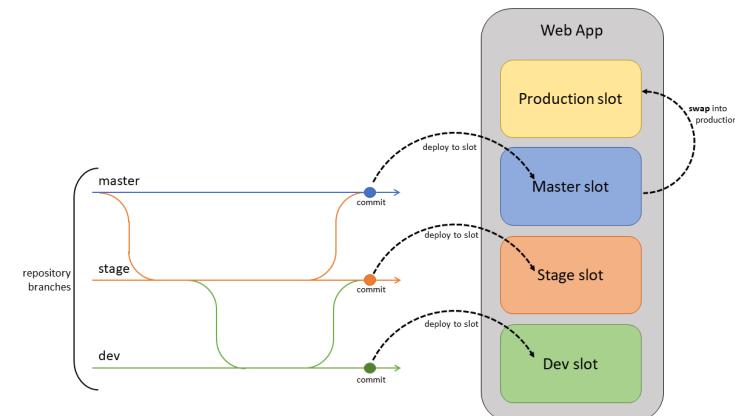
## Use deployment slots

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, validate your changes, and do smoke tests. When you are ready, you can 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 is known as the [Gitflow design](#).) This allows your stakeholders to easily assess and test the deployed branch.

Continuous deployment should never be enabled for your production slot. Instead, your production branch (often main) should be deployed onto a non-production slot. When you are ready to release the base branch, swap it into the production slot. Swapping into production—instead of deploying to production—prevents downtime and allows you to roll back the changes by swapping again.

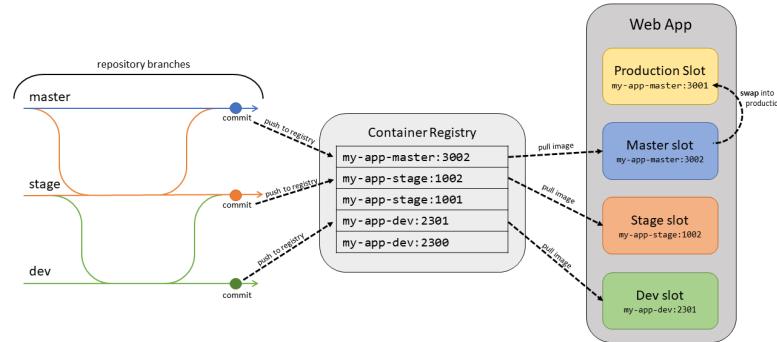


### Continuously deploy containers

For custom containers from Docker 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.

For each branch you want to deploy to a slot, set up automation to do the following on each commit to the branch.

- 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.
- 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.
- 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.



There are examples below for common automation frameworks.

#### Use Azure DevOps

App Service has [built-in continuous delivery](#) for containers through the Deployment Center. Navigate to your app in the [Azure portal](#) and select **Deployment Center** under **Deployment**. Follow the instructions to select your repository and branch. This will configure a DevOps build and release pipeline to automatically build, tag, and deploy your container when new commits are pushed to your selected branch.

#### Use GitHub Actions

You can also automate your container deployment [with GitHub Actions](#). The workflow file below will build and tag the container with the commit ID, push it to a container registry, and update the specified site slot with the new image tag.

```

name: Build and deploy a container image to Azure Web Apps

on:
 push:
 branches:
 - <your-branch-name>

jobs:
 build-and-deploy:
 runs-on: ubuntu-latest

 steps:
 - uses: actions/checkout@main

 - name: Authenticate using a Service Principal
 uses: azure/actions/login@v1
 with:
 creds: ${{ secrets.AZURE_SP }}

 - uses: azure/container-actions/docker-login@v1
 with:
 username: ${{ secrets.DOCKER_USERNAME }}
 password: ${{ secrets.DOCKER_PASSWORD }}

 - name: Build and push the image tagged with the git commit hash
 run: |
 docker build . -t contoso/demo:${{ github.sha }}
 docker push contoso/demo:${{ github.sha }}

 - name: Update image tag on the Azure Web App
 uses: azure/webapps-container-deploy@v1
 with:
 app-name: '<your-webapp-name>'
 slot-name: '<your-slot-name>'
 images: 'contoso/demo:${{ github.sha }}'
```

#### Use other automation providers

The steps listed earlier apply to other automation utilities such as CircleCI or Travis CI. However, you need to use the Azure CLI to update the deployment slots with new image tags in the final step. To use the Azure CLI in your automation script, generate a Service Principal using the following command.

```
az ad sp create-for-rbac --name "myServicePrincipal" --role contributor \
--scopes /subscriptions/{subscription}/resourceGroups/{resource-group} \
--sdk-auth
```

In your script, log in using `az login --service-principal`, providing the principal's information. You can then use `az webapp config container set` to set the container name, tag, registry URL, and registry password. Below are some helpful links for you to construct your container CI process.

- [How to log into the Azure CLI on Circle CI](#)

## Language-Specific Considerations

### Java

Use the Kudu [zipdeploy](#)/ API for deploying JAR applications, and [wardeploy](#)/ for WAR apps. If you are using Jenkins, you can use those APIs directly in your deployment phase. For more information, see [this article](#).

### Node

By default, Kudu executes the build steps for your Node application (`npm install`). If you are using a build service such as Azure DevOps, then the Kudu build is unnecessary. To disable the Kudu build, create an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, with a value of `false`.

### .NET

By default, Kudu executes the build steps for your .NET application (`dotnet build`). If you are using a build service such as Azure DevOps, then the Kudu build is unnecessary. To disable the Kudu build, create an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, with a value of `false`.

## Other Deployment Considerations

### Local Cache

Azure App Service content is stored on Azure Storage and is surfaced up in a durable manner as a content share. However, some apps just need a high-performance, read-only content store that they can run with high availability. These apps can benefit from using [local cache](#). Local cache is not recommended for content management sites such as WordPress.

Always use local cache in conjunction with [deployment slots](#) to prevent downtime. See [this section](#) for information on using these features together.

### High CPU or Memory

If your App Service Plan is using over 90% of available CPU or memory, the underlying virtual machine may have trouble processing your deployment. When this happens, temporarily scale up your instance count to perform the deployment. Once the deployment has finished, you can return the instance count to its previous value.

For more information on best practices, visit [App Service Diagnostics](#) to find out actionable best practices specific to your resource.

- Navigate to your Web App in the [Azure portal](#).
- Click on **Diagnose and solve problems** in the left navigation, which opens App Service Diagnostics.
- Choose **Best Practices** homepage tile.
- Click **Best Practices for Availability & Performance** or **Best Practices for Optimal Configuration** to view the current state of your app in regards to these best practices.

You can also use this link to directly open App Service Diagnostics for your resource:

[https://ms.portal.azure.com/?websitesextension\\_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Web/sites/{siteName}/diagnosticLogs](https://ms.portal.azure.com/?websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Web/sites/{siteName}/diagnosticLogs)

## More resources

[Environment variables and app settings reference](#)

# App Service, Functions, and Logic Apps on Azure Arc (Preview)

11/2/2021 • 7 minutes to read • [Edit Online](#)

You can run App Service, Functions, and Logic Apps on an Azure Arc-enabled Kubernetes cluster. The Kubernetes cluster can be on-premises or hosted in a third-party cloud. This approach lets app developers take advantage of the features of App Service. At the same time, it lets their IT administrators maintain corporate compliance by hosting the App Service apps on internal infrastructure. It also lets other IT operators safeguard their prior investments in other cloud providers by running App Service on existing Kubernetes clusters.

## NOTE

To learn how to set up your Kubernetes cluster for App Service, Functions, and Logic Apps, see [Create an App Service Kubernetes environment \(Preview\)](#).

In most cases, app developers need to know nothing more than how to deploy to the correct Azure region that represents the deployed Kubernetes environment. For operators who provide the environment and maintain the underlying Kubernetes infrastructure, you need to be aware of the following Azure resources:

- The connected cluster, which is an Azure projection of your Kubernetes infrastructure. For more information, see [What is Azure Arc-enabled Kubernetes?](#).
- A cluster extension, which is a sub-resource of the connected cluster resource. The App Service extension [installs the required pods into your connected cluster](#). For more information about cluster extensions, see [Cluster extensions on Azure Arc-enabled Kubernetes](#).
- A custom location, which bundles together a group of extensions and maps them to a namespace for created resources. For more information, see [Custom locations on top of Azure Arc-enabled Kubernetes](#).
- An App Service Kubernetes environment, which enables configuration common across apps but not related to cluster operations. Conceptually, it's deployed into the custom location resource, and app developers create apps into this environment. This is described in greater detail in [App Service Kubernetes environment](#).

## Public preview limitations

The following public preview limitations apply to App Service Kubernetes environments. They will be updated as changes are made available.

LIMITATION	DETAILS
Supported Azure regions	East US, West Europe
Cluster networking requirement	Must support <code>LoadBalancer</code> service type and provide a publicly addressable static IP
Cluster storage requirement	Must have cluster attached storage class available for use by the extension to support deployment and build of code-based apps where applicable
Feature: Networking	<a href="#">Not available (rely on cluster networking)</a>

LIMITATION	DETAILS
Feature: Managed identities	Not available
Feature: Key vault references	Not available (depends on managed identities)
Feature: Pull images from ACR with managed identity	Not available (depends on managed identities)
Feature: In-portal editing for Functions and Logic Apps	Not available
Feature: FTP publishing	Not available
Logs	Log Analytics must be configured with cluster extension; not per-site

## Pods created by the App Service extension

When the App Service extension is installed on the Azure Arc-enabled Kubernetes cluster, you see several pods created in the release namespace that was specified. These pods enable your Kubernetes cluster to be an extension of the `Microsoft.Web` resource provider in Azure and support the management and operation of your apps. Optionally, you can choose to have the extension install [KEDA](#) for event-driven scaling.

The following table describes the role of each pod that is created by default:

POD	DESCRIPTION
<code>&lt;extensionName&gt;-k8se-app-controller</code>	The core operator pod that creates resources on the cluster and maintains the state of components.
<code>&lt;extensionName&gt;-k8se-envoy</code>	A front-end proxy layer for all data-plane requests. It routes the inbound traffic to the correct apps.
<code>&lt;extensionName&gt;-k8se-activator</code>	An alternative routing destination to help with apps that have scaled to zero while the system gets the first instance available.
<code>&lt;extensionName&gt;-k8se-build-service</code>	Supports deployment operations and serves the <a href="#">Advanced tools feature</a> .
<code>&lt;extensionName&gt;-k8se-http-scaler</code>	Monitors inbound request volume in order to provide scaling information to <a href="#">KEDA</a> .
<code>&lt;extensionName&gt;-k8se-img-cacher</code>	Pulls placeholder and app images into a local cache on the node.
<code>&lt;extensionName&gt;-k8se-log-processor</code>	Gathers logs from apps and other components and sends them to Log Analytics.
<code>placeholder-azure-functions-*</code>	Used to speed up cold starts for Azure Functions.

## App Service Kubernetes environment

The App Service Kubernetes environment resource is required before apps may be created. It enables configuration common to apps in the custom location, such as the default DNS suffix.

Only one Kubernetes environment resource may be created in a custom location. In most cases, a developer who creates and deploys apps doesn't need to be directly aware of the resource. It can be directly inferred from the provided custom location ID. However, when defining Azure Resource Manager templates, any plan resource needs to reference the resource ID of the environment directly. The custom location values of the plan and the specified environment must match.

## FAQ for App Service, Functions, and Logic Apps on Azure Arc (Preview)

- [How much does it cost?](#)
- [Are both Windows and Linux apps supported?](#)
- [Which built-in application stacks are supported?](#)
- [Are all app deployment types supported?](#)
- [Which App Service features are supported?](#)
- [Are networking features supported?](#)
- [Are managed identities supported?](#)
- [Are there any scaling limits?](#)
- [What logs are collected?](#)
- [What do I do if I see a provider registration error?](#)
- [Can I deploy the Application services extension on an ARM64 based cluster?](#)

### **How much does it cost?**

App Service on Azure Arc is free during the public preview.

### **Are both Windows and Linux apps supported?**

Only Linux-based apps are supported, both code and custom containers. Windows apps are not supported.

### **Which built-in application stacks are supported?**

All built-in Linux stacks are supported.

### **Are all app deployment types supported?**

FTP deployment is not supported. Currently `az webapp up` is also not supported. Other deployment methods are supported, including Git, ZIP, CI/CD, Visual Studio, and Visual Studio Code.

### **Which App Service features are supported?**

During the preview period, certain App Service features are being validated. When they're supported, their left navigation options in the Azure portal will be activated. Features that are not yet supported remain grayed out.

### **Are networking features supported?**

No. Networking features such as hybrid connections, Virtual Network integration, or IP restrictions, are not supported. Networking should be handled directly in the networking rules in the Kubernetes cluster itself.

### **Are managed identities supported?**

No. Apps cannot be assigned managed identities when running in Azure Arc. If your app needs an identity for working with another Azure resource, consider using an [application service principal](#) instead.

### **Are there any scaling limits?**

All applications deployed with Azure App Service on Kubernetes with Azure Arc are able to scale within the limits of the underlying Kubernetes cluster. If the underlying Kubernetes Cluster runs out of available compute resources (CPU and memory primarily), then applications will only be able to scale to the number of instances of the application that Kubernetes can schedule with available resource.

### **What logs are collected?**

Logs for both system components and your applications are written to standard output. Both log types can be collected for analysis using standard Kubernetes tools. You can also configure the App Service cluster extension with a [Log Analytics workspace](#), and it will send all logs to that workspace.

By default, logs from system components are sent to the Azure team. Application logs are not sent. You can prevent these logs from being transferred by setting `logProcessor.enabled=false` as an extension configuration setting. This will also disable forwarding of application to your Log Analytics workspace. Disabling the log processor may impact time needed for any support cases, and you will be asked to collect logs from standard output through some other means.

### What do I do if I see a provider registration error?

When creating a Kubernetes environment resource, some subscriptions may see a "No registered resource provider found" error. The error details may include a set of locations and api versions that are considered valid. If this happens, it may be that the subscription needs to be re-registered with the Microsoft.Web provider, an operation which has no impact on existing applications or APIs. To re-register, use the Azure CLI to run

```
az provider register --namespace Microsoft.Web --wait
```

 Then re-attempt the Kubernetes environment command.

### Can I deploy the Application services extension on an ARM64 based cluster?

ARM64 based clusters are not supported at this time.

## Extension Release Notes

### Application services extension v 0.9.0 (May 2021)

- Initial public preview release of Application services extension.
- Support for code and container-based deployments of Web, Function, and Logic Applications.
- Web application runtime support - .NET 3.1 and 5.0; Node JS 12 and 14; Python 3.6, 3.7, and 3.8; PHP 7.3 and 7.4; Ruby 2.5, 2.5.5, 2.6, and 2.6.2; Java SE 8u232, 8u242, 8u252, 11.05, 11.06 and 11.07; Tomcat 8.5, 8.5.41, 8.5.53, 8.5.57, 9.0, 9.0.20, 9.0.33, and 9.0.37.

### Application services extension v 0.10.0 (November 2021)

If your extension was in the stable version and auto-upgrade-minor-version is set to enabled the extension will upgrade automatically. To manually upgrade the extension to the latest version of the extension you can run the command below

- Removed requirement for pre-assigned Static IP Address required for assignment to the Envoy endpoint
- Upgrade Keda to v2.4.0
- Upgrade Envoy to v1.19.0
- Upgrade Azure Function runtime to v3.3.1
- Set default replica count of App Controller and Envoy Controller to 2 to add further stability

If your extension was in the stable version and auto-upgrade-minor-version is set to true, the extension will upgrade automatically. To manually upgrade the extension to the latest version, you can run the command below:

```
az k8s-extension update --cluster-type connectedClusters -c <clustername> -g <resource group> -n <extension name> --release-train stable --version 0.10.0
```

## Next steps

[Create an App Service Kubernetes environment \(Preview\)](#)

# Security recommendations for App Service

11/2/2021 • 3 minutes to read • [Edit Online](#)

This article contains security recommendations for Azure App Service. Implementing these recommendations will help you fulfill your security obligations as described in our shared responsibility model and will improve the overall security for your Web App solutions. For more information on what Microsoft does to fulfill service provider responsibilities, read [Azure infrastructure security](#).

## General

RECOMMENDATION	COMMENTS
Stay up-to-date	Use the latest versions of supported platforms, programming languages, protocols, and frameworks.

## Identity and access management

RECOMMENDATION	COMMENTS
Disable anonymous access	Unless you need to support anonymous requests, disable anonymous access. For more information on Azure App Service authentication options, see <a href="#">Authentication and authorization in Azure App Service</a> .
Require authentication	Whenever possible, use the App Service authentication module instead of writing code to handle authentication and authorization. See <a href="#">Authentication and authorization in Azure App Service</a> .
Protect back-end resources with authenticated access	You can either use the user's identity or use an application identity to authenticate to a back-end resource. When you choose to use an application identity use a <a href="#">managed identity</a> .
Require client certificate authentication	Client certificate authentication improves security by only allowing connections from clients that can authenticate using certificates that you provide.

## Data protection

RECOMMENDATION	COMMENTS
Redirect HTTP to HTTPS	By default, clients can connect to web apps by using both HTTP or HTTPS. We recommend redirecting HTTP to HTTPS because HTTPS uses the SSL/TLS protocol to provide a secure connection, which is both encrypted and authenticated.

RECOMMENDATION	COMMENTS
Encrypt communication to Azure resources	When your app connects to Azure resources, such as <a href="#">SQL Database</a> or <a href="#">Azure Storage</a> , the connection stays in Azure. Since the connection goes through the shared networking in Azure, you should always encrypt all communication.
Require the latest TLS version possible	Since 2018 new Azure App Service apps use TLS 1.2. Newer versions of TLS include security improvements over older protocol versions.
Use FTPS	App Service supports both FTP and FTPS for deploying your files. Use FTPS instead of FTP when possible. When one or both of these protocols are not in use, you should <a href="#">disable them</a> .
Secure application data	Don't store application secrets, such as database credentials, API tokens, or private keys in your code or configuration files. The commonly accepted approach is to access them as <a href="#">environment variables</a> using the standard pattern in your language of choice. In Azure App Service, you can define environment variables through <a href="#">app settings</a> and <a href="#">connection strings</a> . App settings and connection strings are stored encrypted in Azure. The app settings are decrypted only before being injected into your app's process memory when the app starts. The encryption keys are rotated regularly. Alternatively, you can integrate your Azure App Service app with <a href="#">Azure Key Vault</a> for advanced secrets management. By <a href="#">accessing the Key Vault with a managed identity</a> , your App Service app can securely access the secrets you need.

## Networking

RECOMMENDATION	COMMENTS
Use static IP restrictions	Azure App Service on Windows lets you define a list of IP addresses that are allowed to access your app. The allowed list can include individual IP addresses or a range of IP addresses defined by a subnet mask. For more information, see <a href="#">Azure App Service Static IP Restrictions</a> .
Use the isolated pricing tier	Except for the isolated pricing tier, all tiers run your apps on the shared network infrastructure in Azure App Service. The isolated tier gives you complete network isolation by running your apps inside a dedicated <a href="#">App Service environment</a> . An App Service environment runs in your own instance of <a href="#">Azure Virtual Network</a> .
Use secure connections when accessing on-premises resources	You can use <a href="#">Hybrid connections</a> , <a href="#">Virtual Network integration</a> , or <a href="#">App Service environment's</a> to connect to on-premises resources.
Limit exposure to inbound network traffic	Network security groups allow you to restrict network access and control the number of exposed endpoints. For more information, see <a href="#">How To Control Inbound Traffic to an App Service Environment</a> .

## Monitoring

RECOMMENDATION	COMMENTS
Use Azure Security Center's Azure Defender for App Service	<p>Azure Defender for App Service is natively integrated with Azure App Service. Security Center assesses the resources covered by your App Service plan and generates security recommendations based on its findings. Use the detailed instructions in <a href="#">these recommendations</a> to harden your App Service resources. Azure Defender also provides threat protection and can detect a multitude of threats covering almost the complete list of MITRE ATT&amp;CK tactics from pre-attack to command and control. For a full list of the Azure App Service alerts, see <a href="#">Azure Defender for App Service alerts</a>.</p>

## Next steps

Check with your application provider to see if there are additional security requirements. For more information on developing secure applications, see [Secure Development Documentation](#).

# Authentication and authorization in Azure App Service and Azure Functions

11/2/2021 • 8 minutes to read • [Edit Online](#)

Azure App Service provides built-in authentication and authorization capabilities (sometimes referred to as "Easy Auth"), so you can sign in users and access data by writing minimal or no code in your web app, RESTful API, and mobile back end, and also [Azure Functions](#). This article describes how App Service helps simplify authentication and authorization for your app.

## Why use the built-in authentication?

You're not required to use this feature for authentication and authorization. You can use the bundled security features in your web framework of choice, or you can write your own utilities. However, you will need to ensure that your solution stays up to date with the latest security, protocol, and browser updates.

Implementing a secure solution for authentication (signing-in users) and authorization (providing access to secure data) can take significant effort. You must make sure to follow industry best practices and standards, and keep your implementation up to date. 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	<code>/auth/login/aad</code>	<a href="#">App Service Microsoft Identity Platform login</a>
Facebook	<code>/auth/login/facebook</code>	<a href="#">App Service Facebook login</a>
Google	<code>/auth/login/google</code>	<a href="#">App Service Google login</a>
Twitter	<code>/auth/login/twitter</code>	<a href="#">App Service Twitter login</a>
Any <a href="#">OpenID Connect</a> provider (preview)	<code>/auth/login/&lt;providerName&gt;</code>	<a href="#">App Service OpenID Connect login</a>

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.

# Considerations for using built-in authentication

Enabling this feature will cause all requests to your application to be automatically redirected to HTTPS, regardless of the App Service configuration setting to enforce HTTPS. You can disable this with the `requireHttps` setting in the V2 configuration. However, we do recommend sticking with HTTPS, and you should ensure no security tokens ever get transmitted over non-secure HTTP connections.

App Service can be used for authentication with or without restricting access to your site content and APIs. To restrict app access only to authenticated users, set **Action to take when request is not authenticated** to log in with one of the configured identity providers. To authenticate but not restrict access, set **Action to take when request is not authenticated** to "Allow anonymous requests (no action)."

## NOTE

You should give each app registration its own permission and consent. Avoid permission sharing between environments by using separate app registrations for separate deployment slots. When testing new code, this practice can help prevent issues from affecting the production app.

## How it works

[Feature architecture](#)

[Authentication flow](#)

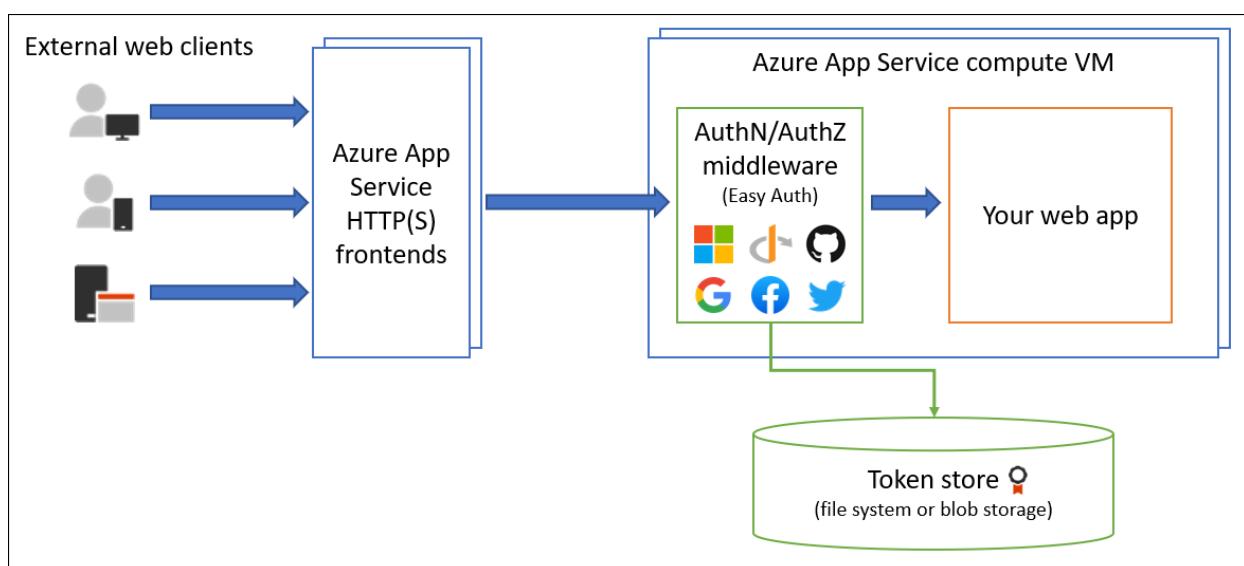
[Authorization behavior](#)

[Token store](#)

[Logging and tracing](#)

### Feature architecture

The authentication and authorization middleware component is a feature of the platform that runs on the same VM as your application. When it's enabled, every incoming HTTP request passes through it before being handled by your application.



The platform middleware handles several things for your app:

- Authenticates users and clients with the specified identity provider(s)
- Validates, stores, and refreshes OAuth tokens issued by the configured identity provider(s)
- Manages the authenticated session

- Injects identity information into HTTP request headers

The module runs separately from your application code and can be configured using Azure Resource Manager settings or using a [configuration file](#). No SDKs, specific programming languages, or changes to your application code are required.

#### **Feature architecture on Windows (non-container deployment)**

The authentication and authorization module runs as a native [IIS module](#) in the same sandbox as your application. When it's enabled, every incoming HTTP request passes through it before being handled by your application.

#### **Feature architecture on Linux and containers**

The authentication and authorization module runs in a separate container, isolated from your application code. Using what's known as the [Ambassador pattern](#), it interacts with the incoming traffic to perform similar functionality as on Windows. Because it does not run in-process, no direct integration with specific language frameworks is possible; however, the relevant information that your app needs is passed through using request headers as explained below.

### **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*. This case applies to browser apps. It also applies to native apps that sign users in using the Mobile Apps client SDK because the SDK opens a web view to sign users in with App Service authentication.
- 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 case applies to REST APIs, [Azure Functions](#), and JavaScript browser clients, as well as browser apps that need more flexibility in the sign-in process. It also applies to native mobile apps that sign users in using the provider's SDK.

Calls from a trusted browser app in App Service to another REST API in App Service or [Azure Functions](#) can be authenticated using the server-directed flow. For more information, see [Customize sign-ins and sign-outs](#).

The table below shows the steps of the authentication flow.

STEP	WITHOUT PROVIDER SDK	WITH PROVIDER SDK
1. Sign user in	Redirects client to <code>/.auth/login/&lt;provider&gt;</code>	Client code signs user in directly with provider's SDK and receives an authentication token. For information, see the provider's documentation.
2. Post-authentication	Provider redirects client to <code>/.auth/login/&lt;provider&gt;/callback</code>	Client code <a href="#">posts token from provider</a> to <code>/.auth/login/&lt;provider&gt;</code> for validation.
3. Establish authenticated session	App Service adds authenticated cookie to response.	App Service returns its own authentication token to client code.

STEP	WITHOUT PROVIDER SDK	WITH PROVIDER SDK
4. Serve authenticated content	Client includes authentication cookie in subsequent requests (automatically handled by browser).	Client code presents authentication token in <code>X-ZUMO-AUTH</code> 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 incoming request is not authenticated. The following headings describe the options.

### 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. For example, it lets you [present multiple sign-in providers](#) to your users. However, you must write code.

### 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.

With this option, you don't need to write any authentication code in your app. Finer authorization, such as role-specific authorization, can be handled by inspecting the user's claims (see [Access user claims](#)).

#### 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.

#### NOTE

By default, any user in your Azure AD tenant can request a token for your application from Azure AD. You can [configure the application in Azure AD](#) if you want to restrict access to your app to a defined set of users.

## Token store

App Service provides a built-in token store, which is a repository of tokens that are associated with the users of your web apps, APIs, or native mobile apps. When you enable authentication with any provider, this token store is immediately available to your app. If your application code needs to access data from these providers on the user's behalf, such as:

- post to the authenticated user's Facebook timeline
- read the user's corporate data using the Microsoft Graph API

You typically must write code to collect, store, and refresh these tokens in your application. With the token store, you just [retrieve the tokens](#) when you need them and [tell App Service to refresh them](#) when they become invalid.

The ID tokens, access tokens, and refresh tokens are cached for the authenticated session, and they're accessible only by the associated user.

If you don't need to work with tokens in your app, you can disable the token store in your app's **Authentication / Authorization** page.

### Logging and tracing

If you [enable application logging](#), you will see authentication and authorization traces directly in your log files. If you see an authentication error that you didn't expect, you can conveniently find all the details by looking in your existing application logs. If you enable [failed request tracing](#), you can see exactly what role the authentication and authorization module may have played in a failed request. In the trace logs, look for references to a module named `EasyAuthModule_32/64`.

## More resources

- [How-To: Configure your App Service or Azure Functions app to use Azure AD login](#)
- [Customize sign-ins and sign-outs](#)
- [Work with OAuth tokens and sessions](#)
- [Access user and application claims](#)
- [File-based configuration](#)

Samples:

- [Tutorial: Add authentication to your web app running on Azure App Service](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service \(Windows or Linux\)](#)
- [.NET Core integration of Azure AppService EasyAuth \(3rd party\)](#)
- [Getting Azure App Service authentication working with .NET Core \(3rd party\)](#)

# OS and runtime patching in Azure App Service

11/2/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to get certain version information regarding the OS or software in [App Service](#).

App Service is a Platform-as-a-Service, which means that the OS and application stack are managed for you by Azure; you only manage your application and its data. More control over the OS and application stack is available you in [Azure Virtual Machines](#). With that in mind, it is nevertheless helpful for you as an App Service user to know more information, such as:

- How and when are OS updates applied?
- How is App Service patched against significant vulnerabilities (such as zero-day)?
- Which OS and runtime versions are running your apps?

For security reasons, certain specifics of security information are not published. However, the article aims to alleviate concerns by maximizing transparency on the process, and how you can stay up-to-date on security-related announcements or runtime updates.

## How and when are OS updates applied?

Azure manages OS patching on two levels, the physical servers and the guest virtual machines (VMs) that run the App Service resources. Both are updated monthly, which aligns to the monthly [Patch Tuesday](#) schedule. These updates are applied automatically, in a way that guarantees the high-availability SLA of Azure services.

For detailed information on how updates are applied, see [Demystifying the magic behind App Service OS updates](#).

## How does Azure deal with significant vulnerabilities?

When severe vulnerabilities require immediate patching, such as [zero-day vulnerabilities](#), the high-priority updates are handled on a case-by-case basis.

Stay current with critical security announcements in Azure by visiting [Azure Security Blog](#).

## When are supported language runtimes updated, added, or deprecated?

New stable versions of supported language runtimes (major, minor, or patch) are periodically added to App Service instances. Some updates overwrite the existing installation, while others are installed side by side with existing versions. An overwrite installation means that your app automatically runs on the updated runtime. A side-by-side installation means you must manually migrate your app to take advantage of a new runtime version. For more information, see one of the subsections.

### NOTE

Information here applies to language runtimes that are built into an App Service app. A custom runtime you upload to App Service, for example, remains unchanged unless you manually upgrade it.

### New patch updates

Patch updates to .NET, PHP, Java SDK, or Tomcat version are applied automatically by overwriting the existing

installation with the latest version. Node.js patch updates are installed side by side with the existing versions (similar to major and minor versions in the next section). New Python patch versions can be installed manually through [site extensions](#), side by side with the built-in Python installations.

## New major and minor versions

When a new major or minor version is added, it is installed side by side with the existing versions. You can manually upgrade your app to the new version. If you configured the runtime version in a configuration file (such as `web.config` and `package.json`), you need to upgrade with the same method. If you used an App Service setting to configure your runtime version, you can change it in the [Azure portal](#) or by running an [Azure CLI](#) command in the [Cloud Shell](#), as shown in the following examples:

```
az webapp config set --net-framework-version v4.7 --resource-group <groupname> --name <appname>
az webapp config set --php-version 7.0 --resource-group <groupname> --name <appname>
az webapp config appsettings set --settings WEBSITE_NODE_DEFAULT_VERSION=8.9.3 --resource-group <groupname>
--name <appname>
az webapp config set --python-version 3.8 --resource-group <groupname> --name <appname>
az webapp config set --java-version 1.8 --java-container Tomcat --java-container-version 9.0 --resource-
group <groupname> --name <appname>
```

## How can I query OS and runtime update status on my instances?

While critical OS information is locked down from access (see [Operating system functionality on Azure App Service](#)), the [Kudu console](#) enables you to query your App Service instance regarding the OS version and runtime versions.

The following table shows how to the versions of Windows and of the language runtime that are running your apps:

INFORMATION	WHERE TO FIND IT
Windows version	See <a href="https://&lt;appname&gt;.scm.azurewebsites.net/Env.cshtml">https://&lt;appname&gt;.scm.azurewebsites.net/Env.cshtml</a> (under System info)
.NET version	At <a href="https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole">https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole</a> , run the following command in the command prompt: <code>reg query "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\NET Framework Setup\NDP\v4\Full"</code>
.NET Core version	At <a href="https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole">https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole</a> , run the following command in the command prompt: <code>dotnet --version</code>
PHP version	At <a href="https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole">https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole</a> , run the following command in the command prompt: <code>php --version</code>
Default Node.js version	In the <a href="#">Cloud Shell</a> , run the following command: <code>az webapp config appsettings list --resource-group &lt;groupname&gt; --name &lt;appname&gt; --query "[? name=='WEBSITE_NODE_DEFAULT_VERSION']"</code>

INFORMATION	WHERE TO FIND IT
Python version	At <code>https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole</code> , run the following command in the command prompt: <code>python --version</code>
Java version	At <code>https://&lt;appname&gt;.scm.azurewebsites.net/DebugConsole</code> , run the following command in the command prompt: <code>java -version</code>

#### NOTE

Access to registry location

`HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Component Based Servicing\Packages`, where information on "KB" patches is stored, is locked down.

## More resources

[Trust Center: Security](#)

[64 bit ASP.NET Core on Azure App Service](#)

# Azure Policy Regulatory Compliance controls for Azure App Service

11/2/2021 • 79 minutes to read • [Edit Online](#)

Regulatory Compliance in Azure Policy provides Microsoft created and managed initiative definitions, known as **built-ins**, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure App Service. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

## IMPORTANT

Each control below is associated with one or more Azure Policy definitions. These policies may help you [assess compliance](#) with the control; however, there often is not a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves; this doesn't ensure you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards may change over time.

## Australian Government ISM PROTECTED

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Australian Government ISM PROTECTED](#). For more information about this compliance standard, see [Australian Government ISM PROTECTED](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Guidelines for Cryptography - Transport Layer Security	1139	Using Transport Layer Security - 1139	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
Guidelines for Cryptography - Transport Layer Security	1139	Using Transport Layer Security - 1139	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
Guidelines for Cryptography - Transport Layer Security	1139	Using Transport Layer Security - 1139	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
Guidelines for System Management - System administration	1386	Restriction of management traffic flows - 1386	<a href="#">Remote debugging should be turned off for API Apps</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Guidelines for System Management - System administration	1386	Restriction of management traffic flows - 1386	Remote debugging should be turned off for Function Apps	1.0.0
Guidelines for System Management - System administration	1386	Restriction of management traffic flows - 1386	Remote debugging should be turned off for Web Applications	1.0.0
Guidelines for Software Development - Web application development	1424	Web browser-based security controls - 1424	CORS should not allow every resource to access your Web Applications	1.0.0
Guidelines for Software Development - Web application development	1552	Web application interactions - 1552	API App should only be accessible over HTTPS	1.0.0
Guidelines for Software Development - Web application development	1552	Web application interactions - 1552	Function App should only be accessible over HTTPS	1.0.0
Guidelines for Software Development - Web application development	1552	Web application interactions - 1552	Web Application should only be accessible over HTTPS	1.0.0

## Azure Security Benchmark

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Identity Management	IM-1	Standardize Azure Active Directory as the central identity and authentication system	Managed identity should be used in your API App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Identity Management	IM-1	Standardize Azure Active Directory as the central identity and authentication system	Managed identity should be used in your Function App	2.0.0
Identity Management	IM-1	Standardize Azure Active Directory as the central identity and authentication system	Managed identity should be used in your Web App	2.0.0
Identity Management	IM-2	Manage application identities securely and automatically	Managed identity should be used in your API App	2.0.0
Identity Management	IM-2	Manage application identities securely and automatically	Managed identity should be used in your Function App	2.0.0
Identity Management	IM-2	Manage application identities securely and automatically	Managed identity should be used in your Web App	2.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	API App should only be accessible over HTTPS	1.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	FTPS only should be required in your API App	2.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	FTPS only should be required in your Function App	2.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	FTPS should be required in your Web App	2.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	Function App should only be accessible over HTTPS	1.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	Latest TLS version should be used in your API App	1.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	Latest TLS version should be used in your Function App	1.0.0
Data Protection	DP-4	Encrypt sensitive information in transit	Latest TLS version should be used in your Web App	1.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Data Protection	DP-4	Encrypt sensitive information in transit	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
Logging and Threat Detection	LT-4	Enable logging for Azure resources	<a href="#">Diagnostic logs in App Services should be enabled</a>	2.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">CORS should not allow every resource to access your API App</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">CORS should not allow every resource to access your Function Apps</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">CORS should not allow every resource to access your Web Applications</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">Function apps should have 'Client Certificates (Incoming client certificates)' enabled</a>	1.0.1
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">Remote debugging should be turned off for API Apps</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">Remote debugging should be turned off for Function Apps</a>	1.0.0
Posture and Vulnerability Management	PV-2	Sustain secure configurations for Azure services	<a href="#">Remote debugging should be turned off for Web Applications</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
Posture and Vulnerability Management	PV-7	Rapidly and automatically remediate software vulnerabilities	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0

## Azure Security Benchmark v1

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
--------	------------	---------------	-----------------------	-------------------------

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Security	1.1	Protect resources using Network Security Groups or Azure Firewall on your Virtual Network	App Service should use a virtual network service endpoint	1.0.0
Network Security	1.3	Protect critical web applications	CORS should not allow every resource to access your API App	1.0.0
Network Security	1.3	Protect critical web applications	CORS should not allow every resource to access your Function Apps	1.0.0
Network Security	1.3	Protect critical web applications	CORS should not allow every resource to access your Web Applications	1.0.0
Network Security	1.3	Protect critical web applications	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Network Security	1.3	Protect critical web applications	Remote debugging should be turned off for API Apps	1.0.0
Network Security	1.3	Protect critical web applications	Remote debugging should be turned off for Function Apps	1.0.0
Network Security	1.3	Protect critical web applications	Remote debugging should be turned off for Web Applications	1.0.0
Logging and Monitoring	2.3	Enable audit logging for Azure resources	Diagnostic logs in App Services should be enabled	2.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	API App should only be accessible over HTTPS	1.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	FTPS only should be required in your API App	2.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	FTPS only should be required in your Function App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Data Protection	4.4	Encrypt all sensitive information in transit	FTPS should be required in your Web App	2.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Function App should only be accessible over HTTPS	1.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Latest TLS version should be used in your API App	1.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Latest TLS version should be used in your Function App	1.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Latest TLS version should be used in your Web App	1.0.0
Data Protection	4.4	Encrypt all sensitive information in transit	Web Application should only be accessible over HTTPS	1.0.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
Vulnerability Management	5.3	Deploy automated third-party software patch management solution	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
Secure Configuration	7.12	Manage identities securely and automatically	Managed identity should be used in your API App	2.0.0
Secure Configuration	7.12	Manage identities securely and automatically	Managed identity should be used in your Function App	2.0.0
Secure Configuration	7.12	Manage identities securely and automatically	Managed identity should be used in your Web App	2.0.0

## Canada Federal PBMM

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Canada Federal PBMM](#). For more information about this compliance standard, see [Canada Federal PBMM](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC-17(1)	Remote Access   Automated Monitoring / Control	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17(1)	Remote Access   Automated Monitoring / Control	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17(1)	Remote Access   Automated Monitoring / Control	Remote debugging should be turned off for Web Applications	1.0.0
System and Communications Protection	SC-8(1)	Transmission Confidentiality and Integrity   Cryptographic or Alternate Physical Protection	API App should only be accessible over HTTPS	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8(1)	Transmission Confidentiality and Integrity   Cryptographic or Alternate Physical Protection	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8(1)	Transmission Confidentiality and Integrity   Cryptographic or Alternate Physical Protection	Web Application should only be accessible over HTTPS	1.0.0

## CIS Microsoft Azure Foundations Benchmark 1.1.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.1.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
AppService	9.1	Ensure App Service Authentication is set on Azure App Service	Authentication should be enabled on your API app	1.0.0
AppService	9.1	Ensure App Service Authentication is set on Azure App Service	Authentication should be enabled on your Function app	1.0.0
AppService	9.1	Ensure App Service Authentication is set on Azure App Service	Authentication should be enabled on your web app	1.0.0
AppService	9.2	Ensure web app redirects all HTTP traffic to HTTPS in Azure App Service	Web Application should only be accessible over HTTPS	1.0.0
AppService	9.3	Ensure web app is using the latest version of TLS encryption	Latest TLS version should be used in your API App	1.0.0
AppService	9.3	Ensure web app is using the latest version of TLS encryption	Latest TLS version should be used in your Function App	1.0.0
AppService	9.3	Ensure web app is using the latest version of TLS encryption	Latest TLS version should be used in your Web App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
AppService	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	1.0.1
AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Managed identity should be used in your API App	2.0.0
AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Managed identity should be used in your Function App	2.0.0
AppService	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Managed identity should be used in your Web App	2.0.0
AppService	9.7	Ensure that 'PHP version' is the latest, if used to run the web app	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
AppService	9.7	Ensure that 'PHP version' is the latest, if used to run the web app	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
AppService	9.8	Ensure that 'Python version' is the latest, if used to run the web app	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
AppService	9.8	Ensure that 'Python version' is the latest, if used to run the web app	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
AppService	9.8	Ensure that 'Python version' is the latest, if used to run the web app	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
AppService	9.9	Ensure that 'Java version' is the latest, if used to run the web app	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
AppService	9.9	Ensure that 'Java version' is the latest, if used to run the web app	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
AppService	9.9	Ensure that 'Java version' is the latest, if used to run the web app	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
AppService	9.10	Ensure that 'HTTP Version' is the latest, if used to run the web app	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
AppService	9.10	Ensure that 'HTTP Version' is the latest, if used to run the web app	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
AppService	9.10	Ensure that 'HTTP Version' is the latest, if used to run the web app	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0

## CIS Microsoft Azure Foundations Benchmark 1.3.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.3.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Logging and Monitoring	5.3	Ensure that Diagnostic Logs are enabled for all services which support it.	Diagnostic logs in App Services should be enabled	2.0.0
App Service	9.1	Ensure App Service Authentication is set on Azure App Service	Authentication should be enabled on your API app	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
App Service	9.1	Ensure App Service Authentication is set on Azure App Service	<a href="#">Authentication should be enabled on your Function app</a>	1.0.0
App Service	9.1	Ensure App Service Authentication is set on Azure App Service	<a href="#">Authentication should be enabled on your web app</a>	1.0.0
App Service	9.2	Ensure web app redirects all HTTP traffic to HTTPS in Azure App Service	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
App Service	9.3	Ensure web app is using the latest version of TLS encryption	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
App Service	9.3	Ensure web app is using the latest version of TLS encryption	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
App Service	9.3	Ensure web app is using the latest version of TLS encryption	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
App Service	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	<a href="#">Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'</a>	1.0.0
App Service	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	<a href="#">Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'</a>	1.0.0
App Service	9.4	Ensure the web app has 'Client Certificates (Incoming client certificates)' set to 'On'	<a href="#">Function apps should have 'Client Certificates (Incoming client certificates)' enabled</a>	1.0.1
App Service	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	<a href="#">Managed identity should be used in your API App</a>	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
App Service	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Managed identity should be used in your Function App	2.0.0
App Service	9.5	Ensure that Register with Azure Active Directory is enabled on App Service	Managed identity should be used in your Web App	2.0.0
App Service	9.6	Ensure that 'PHP version' is the latest, if used to run the web app	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
App Service	9.6	Ensure that 'PHP version' is the latest, if used to run the web app	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
App Service	9.7	Ensure that 'Python version' is the latest, if used to run the web app	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
App Service	9.7	Ensure that 'Python version' is the latest, if used to run the web app	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
App Service	9.7	Ensure that 'Python version' is the latest, if used to run the web app	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
App Service	9.8	Ensure that 'Java version' is the latest, if used to run the web app	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
App Service	9.8	Ensure that 'Java version' is the latest, if used to run the web app	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
App Service	9.8	Ensure that 'Java version' is the latest, if used to run the web app	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
App Service	9.9	Ensure that 'HTTP Version' is the latest, if used to run the web app	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
App Service	9.9	Ensure that 'HTTP Version' is the latest, if used to run the web app	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
App Service	9.9	Ensure that 'HTTP Version' is the latest, if used to run the web app	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
App Service	9.10	Ensure FTP deployments are disabled	FTPS only should be required in your API App	2.0.0
App Service	9.10	Ensure FTP deployments are disabled	FTPS only should be required in your Function App	2.0.0
App Service	9.10	Ensure FTP deployments are disabled	FTPS should be required in your Web App	2.0.0

## CMMC Level 3

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CMMC Level 3](#). For more information about this compliance standard, see [Cybersecurity Maturity Model Certification \(CMMC\)](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	CORS should not allow every resource to access your API App	1.0.0
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	CORS should not allow every resource to access your Function Apps	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC.1.001	Limit information system access to authorized users, processes acting on behalf of authorized users, and devices (including other information systems).	Remote debugging should be turned off for Web Applications	1.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	API App should only be accessible over HTTPS	1.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	CORS should not allow every resource to access your API App	1.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	CORS should not allow every resource to access your Function Apps	1.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	Function App should only be accessible over HTTPS	1.0.0
Access Control	AC.1.002	Limit information system access to the types of transactions and functions that authorized users are permitted to execute.	Web Application should only be accessible over HTTPS	1.0.0
Access Control	AC.2.013	Monitor and control remote access sessions.	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC.2.013	Monitor and control remote access sessions.	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC.2.013	Monitor and control remote access sessions.	Remote debugging should be turned off for Web Applications	1.0.0
Access Control	AC.2.016	Control the flow of CUI in accordance with approved authorizations.	CORS should not allow every resource to access your API App	1.0.0
Access Control	AC.2.016	Control the flow of CUI in accordance with approved authorizations.	CORS should not allow every resource to access your Function Apps	1.0.0
Audit and Accountability	AU.3.048	Collect audit information (e.g., logs) into one or more central repositories.	Diagnostic logs in App Services should be enabled	2.0.0
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	CORS should not allow every resource to access your API App	1.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	CORS should not allow every resource to access your Function Apps	1.0.0
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	CORS should not allow every resource to access your Web Applications	1.0.0
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	Remote debugging should be turned off for API Apps	1.0.0
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	Remote debugging should be turned off for Function Apps	1.0.0
Configuration Management	CM.3.068	Restrict, disable, or prevent the use of nonessential programs, functions, ports, protocols, and services.	Remote debugging should be turned off for Web Applications	1.0.0
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	API App should only be accessible over HTTPS	1.0.0
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	Function App should only be accessible over HTTPS	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
Identification and Authentication	IA.3.084	Employ replay-resistant authentication mechanisms for network access to privileged and nonprivileged accounts.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Latest TLS version should be used in your Web App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC.1.175	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	<a href="#">CORS should not allow every resource to access your API App</a>	1.0.0
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	<a href="#">CORS should not allow every resource to access your Function Apps</a>	1.0.0
System and Communications Protection	SC.3.183	Deny network communications traffic by default and allow network communications traffic by exception (i.e., deny all, permit by exception).	<a href="#">CORS should not allow every resource to access your Web Applications</a>	1.0.0
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC.3.185	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	Web Application should only be accessible over HTTPS	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC.3.190	Protect the authenticity of communications sessions.	Web Application should only be accessible over HTTPS	1.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Latest TLS version should be used in your API App	1.0.0
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Latest TLS version should be used in your Function App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI.1.210	Identify, report, and correct information and information system flaws in a timely manner.	Latest TLS version should be used in your Web App	1.0.0

## FedRAMP High

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP High](#). For more information about this compliance standard, see [FedRAMP High](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-2	Account Management	Managed identity should be used in your API App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Function App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Web App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your API App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Function App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Web App	2.0.0
Access Control	AC-4	Information Flow Enforcement	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Function Apps	1.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Web Applications	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for Web Applications	1.0.0
Audit and Accountability	AU-6 (4)	Central Review and Analysis	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-6 (5)	Integration / Scanning and Monitoring Capabilities	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-12	Audit Generation	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-12 (1)	System-wide / Time-correlated Audit Trail	Diagnostic logs in App Services should be enabled	2.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your API App	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Web Applications	1.0.0
Configuration Management	CM-6	Configuration Settings	Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Configuration Management	CM-6	Configuration Settings	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Configuration Management	CM-6	Configuration Settings	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	1.0.1
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for API Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Web Applications	1.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Web App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Web App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS only should be required in your API App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">FTPS only should be required in your Function App</a>	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">FTPS should be required in your Web App</a>	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">FTPS only should be required in your API App</a>	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">FTPS only should be required in your Function App</a>	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">FTPS should be required in your Web App</a>	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Web Application should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-28	Protection of Information at Rest	App Service Environment should enable internal encryption	1.0.0
System and Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should enable internal encryption	1.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0

## FedRAMP Moderate

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP Moderate](#). For more information about this compliance standard, see [FedRAMP Moderate](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-2	Account Management	Managed identity should be used in your API App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Function App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Web App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your API App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Function App	2.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Web App	2.0.0
Access Control	AC-4	Information Flow Enforcement	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Web Applications	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for Web Applications	1.0.0
Audit and Accountability	AU-12	Audit Generation	Diagnostic logs in App Services should be enabled	2.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your API App	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Web Applications	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Configuration Management	CM-6	Configuration Settings	Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Configuration Management	CM-6	Configuration Settings	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Configuration Management	CM-6	Configuration Settings	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	1.0.1
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for API Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Web Applications	1.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Web App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Web App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">FTPS only should be required in your API App</a>	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">FTPS only should be required in your Function App</a>	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">FTPS should be required in your Web App</a>	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">FTPS only should be required in your API App</a>	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">FTPS only should be required in your Function App</a>	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	<a href="#">FTPS should be required in your Web App</a>	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Web Application should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-28	Protection of Information at Rest	App Service Environment should enable internal encryption	1.0.0
System and Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should enable internal encryption	1.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0

## HIPAA HITRUST 9.2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - HIPAA HITRUST 9.2](#). For more information about this compliance standard, see [HIPAA HITRUST 9.2](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Remote Diagnostic and Configuration Port Protection	1194.01 2Organizational.2 - 01.l	Ports, services, and similar applications installed on a computer or network systems, which are not specifically required for business functionality, are disabled or removed.	Remote debugging should be turned off for Web Applications	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Remote Diagnostic and Configuration Port Protection	1195.01l3Organizational.1 - 01.l	The organization reviews the information system within every three hundred and sixty-five (365) days to identify and disables unnecessary and non-secure functions, ports, protocols, and/or services.	<a href="#">Remote debugging should be turned off for Function Apps</a>	1.0.0
Remote Diagnostic and Configuration Port Protection	1196.01l3Organizational.24 - 01.l	The organization identifies unauthorized (blacklisted) software on the information system, prevents program execution in accordance with a list of unauthorized (blacklisted) software programs, employs an allow-all, deny-by exception policy to prohibit execution of known unauthorized (blacklisted) software, and reviews and updates the list of unauthorized (blacklisted) software programs annually.	<a href="#">Remote debugging should be turned off for API Apps</a>	1.0.0
Segregation in Networks	0805.01m1Organizational.12 - 01.m	The organization's security gateways (e.g. firewalls) enforce security policies and are configured to filter traffic between domains, block unauthorized access, and are used to maintain segregation between internal wired, internal wireless, and external network segments (e.g., the Internet) including DMZs and enforce access control policies for each of the domains.	<a href="#">App Service should use a virtual network service endpoint</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Segregation in Networks	0806.01m2Organizational.12356 - 01.m	The organizations network is logically and physically segmented with a defined security perimeter and a graduated set of controls, including subnetworks for publicly accessible system components that are logically separated from the internal network, based on organizational requirements; and traffic is controlled based on functionality required and classification of the data/systems based on a risk assessment and their respective security requirements.	<a href="#">App Service should use a virtual network service endpoint</a>	1.0.0
Segregation in Networks	0894.01m2Organizational.7 - 01.m	Networks are segregated from production-level networks when migrating physical servers, applications or data to virtualized servers.	<a href="#">App Service should use a virtual network service endpoint</a>	1.0.0
Network Connection Control	0809.01n2Organizational.1234 - 01.n	Network traffic is controlled in accordance with the organizations access control policy through firewall and other network-related restrictions for each network access point or external telecommunication service's managed interface.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0809.01n2Organizational.1234 - 01.n	Network traffic is controlled in accordance with the organizations access control policy through firewall and other network-related restrictions for each network access point or external telecommunication service's managed interface.	Function App should only be accessible over HTTPS	1.0.0
Network Connection Control	0809.01n2Organizational.1234 - 01.n	Network traffic is controlled in accordance with the organizations access control policy through firewall and other network-related restrictions for each network access point or external telecommunication service's managed interface.	Latest TLS version should be used in your API App	1.0.0
Network Connection Control	0809.01n2Organizational.1234 - 01.n	Network traffic is controlled in accordance with the organizations access control policy through firewall and other network-related restrictions for each network access point or external telecommunication service's managed interface.	Latest TLS version should be used in your Function App	1.0.0
Network Connection Control	0809.01n2Organizational.1234 - 01.n	Network traffic is controlled in accordance with the organizations access control policy through firewall and other network-related restrictions for each network access point or external telecommunication service's managed interface.	Latest TLS version should be used in your Web App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0809.01n2Organizational.1234 - 01.n	Network traffic is controlled in accordance with the organizations access control policy through firewall and other network-related restrictions for each network access point or external telecommunication service's managed interface.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0810.01n2Organizational.5 - 01.n	Transmitted information is secured and, at a minimum, encrypted over open, public networks.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0810.01n2Organizational.5 - 01.n	Transmitted information is secured and, at a minimum, encrypted over open, public networks.	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0810.01n2Organizational.5 - 01.n	Transmitted information is secured and, at a minimum, encrypted over open, public networks.	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
Network Connection Control	0810.01n2Organizational.5 - 01.n	Transmitted information is secured and, at a minimum, encrypted over open, public networks.	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
Network Connection Control	0810.01n2Organizational.5 - 01.n	Transmitted information is secured and, at a minimum, encrypted over open, public networks.	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
Network Connection Control	0810.01n2Organizational.5 - 01.n	Transmitted information is secured and, at a minimum, encrypted over open, public networks.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0811.01n2Organizational.6 - 01.n	Exceptions to the traffic flow policy are documented with a supporting mission/business need, duration of the exception, and reviewed at least annually; traffic flow policy exceptions are removed when no longer supported by an explicit mission/business need.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0811.01n2Organizational.6 - 01.n	Exceptions to the traffic flow policy are documented with a supporting mission/business need, duration of the exception, and reviewed at least annually; traffic flow policy exceptions are removed when no longer supported by an explicit mission/business need.	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0811.01n2Organizational.6 - 01.n	Exceptions to the traffic flow policy are documented with a supporting mission/business need, duration of the exception, and reviewed at least annually; traffic flow policy exceptions are removed when no longer supported by an explicit mission/business need.	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0811.01n2Organizational.6 - 01.n	Exceptions to the traffic flow policy are documented with a supporting mission/business need, duration of the exception, and reviewed at least annually; traffic flow policy exceptions are removed when no longer supported by an explicit mission/business need.	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
Network Connection Control	0811.01n2Organizational.6 - 01.n	Exceptions to the traffic flow policy are documented with a supporting mission/business need, duration of the exception, and reviewed at least annually; traffic flow policy exceptions are removed when no longer supported by an explicit mission/business need.	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
Network Connection Control	0811.01n2Organizational.6 - 01.n	Exceptions to the traffic flow policy are documented with a supporting mission/business need, duration of the exception, and reviewed at least annually; traffic flow policy exceptions are removed when no longer supported by an explicit mission/business need.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0812.01n2Organizational.8 - 01.n	Remote devices establishing a non-remote connection are not allowed to communicate with external (remote) resources.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0812.01n2Organizational.8 - 01.n	Remote devices establishing a non-remote connection are not allowed to communicate with external (remote) resources.	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0812.01n2Organizational.8 - 01.n	Remote devices establishing a non-remote connection are not allowed to communicate with external (remote) resources.	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
Network Connection Control	0812.01n2Organizational.8 - 01.n	Remote devices establishing a non-remote connection are not allowed to communicate with external (remote) resources.	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
Network Connection Control	0812.01n2Organizational.8 - 01.n	Remote devices establishing a non-remote connection are not allowed to communicate with external (remote) resources.	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0
Network Connection Control	0812.01n2Organizational.8 - 01.n	Remote devices establishing a non-remote connection are not allowed to communicate with external (remote) resources.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
Network Connection Control	0814.01n1Organizational.12 - 01.n	The ability of users to connect to the internal network is restricted using a deny-by-default and allow-by-exception policy at managed interfaces according to the access control policy and the requirements of clinical and business applications.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0814.01n1Organizational.12 - 01.n	The ability of users to connect to the internal network is restricted using a deny-by-default and allow-by-exception policy at managed interfaces according to the access control policy and the requirements of clinical and business applications.	Function App should only be accessible over HTTPS	1.0.0
Network Connection Control	0814.01n1Organizational.12 - 01.n	The ability of users to connect to the internal network is restricted using a deny-by-default and allow-by-exception policy at managed interfaces according to the access control policy and the requirements of clinical and business applications.	Latest TLS version should be used in your API App	1.0.0
Network Connection Control	0814.01n1Organizational.12 - 01.n	The ability of users to connect to the internal network is restricted using a deny-by-default and allow-by-exception policy at managed interfaces according to the access control policy and the requirements of clinical and business applications.	Latest TLS version should be used in your Function App	1.0.0
Network Connection Control	0814.01n1Organizational.12 - 01.n	The ability of users to connect to the internal network is restricted using a deny-by-default and allow-by-exception policy at managed interfaces according to the access control policy and the requirements of clinical and business applications.	Latest TLS version should be used in your Web App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Connection Control	0814.01n1Organizational.12 - 01.n	The ability of users to connect to the internal network is restricted using a deny-by-default and allow-by-exception policy at managed interfaces according to the access control policy and the requirements of clinical and business applications.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
Identification of Risks Related to External Parties	1402.05i1Organizational.45 - 05.i	Remote access connections between the organization and external parties are encrypted.	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
Identification of Risks Related to External Parties	1403.05i1Organizational.67 - 05.i	Access granted to external parties is limited to the minimum necessary and granted only for the duration required.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
Identification of Risks Related to External Parties	1404.05i2Organizational.1 - 05.i	Due diligence of the external party includes interviews, document review, checklists, certification reviews (e.g. HITRUST) or other remote means.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
Audit Logging	1209.09aa3System.2 - 09.aa	The information system generates audit records containing the following detailed information: (i) filename accessed; (ii) program or command used to initiate the event; and (iii) source and destination addresses.	<a href="#">Diagnostic logs in App Services should be enabled</a>	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Network Controls	0861.09m2Organizational.67 - 09.m	To identify and authenticate devices on local and/or wide area networks, including wireless networks, the information system uses either a (i) shared known information solution or (ii) an organizational authentication solution, the exact selection and strength of which is dependent on the security categorization of the information system.	<a href="#">App Service should use a virtual network service endpoint</a>	1.0.0
Information Exchange Policies and Procedures	0662.09sCSPOrganizational.2 - 09.s	Cloud service providers use an industry-recognized virtualization platform and standard virtualization formats (e.g., Open Virtualization Format, OVF) to help ensure interoperability, and has documented custom changes made to any hypervisor in use and all solution-specific virtualization hooks available for customer review.	<a href="#">Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'</a>	1.0.0
Information Exchange Policies and Procedures	0901.09s1Organizational.1 - 09.s	The organization formally addresses multiple safeguards before allowing the use of information systems for information exchange.	<a href="#">CORS should not allow every resource to access your Web Applications</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Information Exchange Policies and Procedures	0902.09s2Organizational.13 - 09.s	Remote (external) access to the organization's information assets and access to external information assets (for which the organization has no control) is based on clearly defined terms and conditions.	<a href="#">CORS should not allow every resource to access your Function Apps</a>	1.0.0
Information Exchange Policies and Procedures	0911.09s1Organizational.2 - 09.s	The organization establishes terms and conditions, consistent with any trust relationship established with other organizations owning, operating, and/or maintaining external information systems, allowing authorized individuals to (i) access the information system from external information systems; and (ii) process, store or transmit organization-controlled information using external information systems.	<a href="#">CORS should not allow every resource to access your API App</a>	1.0.0
Information Exchange Policies and Procedures	0912.09s1Organizational.4 - 09.s	Cryptography is used to protect the confidentiality and integrity of remote access sessions to the internal network and to external systems.	<a href="#">Remote debugging should be turned off for Web Applications</a>	1.0.0
Information Exchange Policies and Procedures	0913.09s1Organizational.5 - 09.s	Strong cryptography protocols are used to safeguard covered information during transmission over less trusted / open public networks.	<a href="#">Remote debugging should be turned off for Function Apps</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Information Exchange Policies and Procedures	0914.09s1Organizational.6 - 09.s	The organization ensures that communication protection requirements, including the security of exchanges of information, is the subject of policy development and compliance audits.	Remote debugging should be turned off for API Apps	1.0.0
Information Exchange Policies and Procedures	0915.09s2Organizational.2 - 09.s	The organization limits the use of organization-controlled portable storage media by authorized individuals on external information systems.	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Information Exchange Policies and Procedures	0916.09s2Organizational.4 - 09.s	The information system prohibits remote activation of collaborative computing devices and provides an explicit indication of use to users physically present at the devices.	CORS should not allow every resource to access your Web Applications	1.0.0
Information Exchange Policies and Procedures	0960.09sCSPOrganizational.1 - 09.s	Cloud service providers use secure (e.g., non-clear text and authenticated) standardized network protocols for the import and export of data and to manage the service, and make available a document to consumers (tenants) detailing the relevant interoperability and portability standards that are involved.	CORS should not allow every resource to access your Function Apps	1.0.0
Information Exchange Policies and Procedures	1325.09s1Organizational.3 - 09.s	Personnel are appropriately trained on leading principles and practices for all types of information exchange (oral, paper and electronic).	Remote debugging should be turned off for Function Apps	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
On-line Transactions	0949.09y2Organizational.5 - 09.y	The protocols used for communications are enhanced to address any new vulnerability, and the updated versions of the protocols are adopted as soon as possible.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
On-line Transactions	0949.09y2Organizational.5 - 09.y	The protocols used for communications are enhanced to address any new vulnerability, and the updated versions of the protocols are adopted as soon as possible.	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
On-line Transactions	0949.09y2Organizational.5 - 09.y	The protocols used for communications are enhanced to address any new vulnerability, and the updated versions of the protocols are adopted as soon as possible.	<a href="#">Latest TLS version should be used in your API App</a>	1.0.0
On-line Transactions	0949.09y2Organizational.5 - 09.y	The protocols used for communications are enhanced to address any new vulnerability, and the updated versions of the protocols are adopted as soon as possible.	<a href="#">Latest TLS version should be used in your Function App</a>	1.0.0
On-line Transactions	0949.09y2Organizational.5 - 09.y	The protocols used for communications are enhanced to address any new vulnerability, and the updated versions of the protocols are adopted as soon as possible.	<a href="#">Latest TLS version should be used in your Web App</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
On-line Transactions	0949.09y2Organizational.5 - 09.y	The protocols used for communications are enhanced to address any new vulnerability, and the updated versions of the protocols are adopted as soon as possible.	Web Application should only be accessible over HTTPS	1.0.0

## IRS 1075 September 2016

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - IRS 1075 September 2016](#). For more information about this compliance standard, see [IRS 1075 September 2016](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	9.3.1.4	Information Flow Enforcement (AC-4)	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	9.3.1.12	Remote Access (AC-17)	Remote debugging should be turned off for API Apps	1.0.0
Access Control	9.3.1.12	Remote Access (AC-17)	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	9.3.1.12	Remote Access (AC-17)	Remote debugging should be turned off for Web Applications	1.0.0
System and Communications Protection	9.3.16.6	Transmission Confidentiality and Integrity (SC-8)	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	9.3.16.6	Transmission Confidentiality and Integrity (SC-8)	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	9.3.16.6	Transmission Confidentiality and Integrity (SC-8)	Web Application should only be accessible over HTTPS	1.0.0

## ISO 27001:2013

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - ISO 27001:2013](#). For more information about this compliance standard, see [ISO 27001:2013](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Cryptography	10.1.1	Policy on the use of cryptographic controls	API App should only be accessible over HTTPS	1.0.0
Cryptography	10.1.1	Policy on the use of cryptographic controls	Function App should only be accessible over HTTPS	1.0.0
Cryptography	10.1.1	Policy on the use of cryptographic controls	Web Application should only be accessible over HTTPS	1.0.0

## New Zealand ISM Restricted

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - New Zealand ISM Restricted](#). For more information about this compliance standard, see [New Zealand ISM Restricted](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Software security	SS-2	14.1.8 Developing hardened SOEs	Remote debugging should be turned off for API Apps	1.0.0
Software security	SS-2	14.1.8 Developing hardened SOEs	Remote debugging should be turned off for Function Apps	1.0.0
Software security	SS-2	14.1.8 Developing hardened SOEs	Remote debugging should be turned off for Web Applications	1.0.0
Software security	SS-9	14.5.8 Web applications	API App should only be accessible over HTTPS	1.0.0
Software security	SS-9	14.5.8 Web applications	CORS should not allow every resource to access your API App	1.0.0
Software security	SS-9	14.5.8 Web applications	CORS should not allow every resource to access your Function Apps	1.0.0
Software security	SS-9	14.5.8 Web applications	CORS should not allow every resource to access your Web Applications	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Software security	SS-9	14.5.8 Web applications	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
Software security	SS-9	14.5.8 Web applications	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
Software security	SS-9	14.5.8 Web applications	Function App should only be accessible over HTTPS	1.0.0
Software security	SS-9	14.5.8 Web applications	Web Application should only be accessible over HTTPS	1.0.0
Access Control and Passwords	AC-2	16.1.32 System User Identification	Managed identity should be used in your API App	2.0.0
Access Control and Passwords	AC-2	16.1.32 System User Identification	Managed identity should be used in your Function App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
--------	------------	---------------	--------	----------------

Access Control and Passwords	AC-2	16.1.32 System User Identification	Managed identity should be used in your Web App	2.0.0
Access Control and Passwords	AC-17	16.6.9 Events to be logged	Diagnostic logs in App Services should be enabled	2.0.0
Cryptography	CR-7	17.4.16 Using TLS	FTPS only should be required in your API App	2.0.0
Cryptography	CR-7	17.4.16 Using TLS	FTPS only should be required in your Function App	2.0.0
Cryptography	CR-7	17.4.16 Using TLS	FTPS should be required in your Web App	2.0.0
Cryptography	CR-7	17.4.16 Using TLS	Latest TLS version should be used in your API App	1.0.0
Cryptography	CR-7	17.4.16 Using TLS	Latest TLS version should be used in your Function App	1.0.0
Cryptography	CR-7	17.4.16 Using TLS	Latest TLS version should be used in your Web App	1.0.0

## NIST SP 800-171 R2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-171 R2](#). For more information about this compliance standard, see [NIST SP 800-171 R2](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	Remote debugging should be turned off for API Apps	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	<a href="#">Remote debugging should be turned off for Function Apps</a>	1.0.0
Access Control	3.1.1	Limit system access to authorized users, processes acting on behalf of authorized users, and devices (including other systems).	<a href="#">Remote debugging should be turned off for Web Applications</a>	1.0.0
Access Control	3.1.3	Control the flow of CUI in accordance with approved authorizations.	<a href="#">CORS should not allow every resource to access your Web Applications</a>	1.0.0
Access Control	3.1.12	Monitor and control remote access sessions.	<a href="#">Remote debugging should be turned off for API Apps</a>	1.0.0
Access Control	3.1.12	Monitor and control remote access sessions.	<a href="#">Remote debugging should be turned off for Function Apps</a>	1.0.0
Access Control	3.1.12	Monitor and control remote access sessions.	<a href="#">Remote debugging should be turned off for Web Applications</a>	1.0.0
System and Communications Protection	3.13.1	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	3.13.1	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	3.13.1	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	3.13.1	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	3.13.1	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	Latest TLS version should be used in your Web App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	3.13.1	Monitor, control, and protect communications (i.e., information transmitted or received by organizational systems) at the external boundaries and key internal boundaries of organizational systems.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	<a href="#">API App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	<a href="#">Function App should only be accessible over HTTPS</a>	1.0.0
System and Communications Protection	3.13.8	Implement cryptographic mechanisms to prevent unauthorized disclosure of CUI during transmission unless otherwise protected by alternative physical safeguards.	<a href="#">Web Application should only be accessible over HTTPS</a>	1.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	<a href="#">Ensure that 'HTTP Version' is the latest, if used to run the API app</a>	2.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	<a href="#">Ensure that 'HTTP Version' is the latest, if used to run the Function app</a>	2.0.0

Domain	Control ID	Control Title	Policy	Policy Version
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Latest TLS version should be used in your API App	1.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Latest TLS version should be used in your Function App	1.0.0
System and Information Integrity	3.14.1	Identify, report, and correct system flaws in a timely manner.	Latest TLS version should be used in your Web App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
--------	------------	---------------	--------	----------------

## NIST SP 800-53 Rev. 4

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 4](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 4](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-2	Account Management	Managed identity should be used in your API App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Function App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Web App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your API App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Function App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Web App	2.0.0
Access Control	AC-4	Information Flow Enforcement	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Web Applications	1.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Remote debugging should be turned off for Web Applications	1.0.0
Audit and Accountability	AU-6 (4)	Central Review and Analysis	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-6 (5)	Integration / Scanning and Monitoring Capabilities	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-12	Audit Generation	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-12 (1)	System-wide / Time-correlated Audit Trail	Diagnostic logs in App Services should be enabled	2.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your API App	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Web Applications	1.0.0
Configuration Management	CM-6	Configuration Settings	Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Configuration Management	CM-6	Configuration Settings	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Configuration Management	CM-6	Configuration Settings	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	1.0.1
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for API Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Web Applications	1.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Web App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Web App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS only should be required in your API App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS only should be required in your Function App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS should be required in your Web App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Web Application should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	FTPS only should be required in your API App	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	FTPS only should be required in your Function App	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	FTPS should be required in your Web App	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your Function App	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic or Alternate Physical Protection	Web Application should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-28	Protection of Information at Rest	App Service Environment should enable internal encryption	1.0.0
System and Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should enable internal encryption	1.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software / Firmware	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0

## NIST SP 800-53 Rev. 5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 5](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 5](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-2	Account Management	Managed identity should be used in your API App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Function App	2.0.0
Access Control	AC-2	Account Management	Managed identity should be used in your Web App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your API App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Function App	2.0.0
Access Control	AC-3	Access Enforcement	Managed identity should be used in your Web App	2.0.0

Domain	Control ID	Control Title	Policy	Policy Version
Access Control	AC-4	Information Flow Enforcement	CORS should not allow every resource to access your Web Applications	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17	Remote Access	Remote debugging should be turned off for Web Applications	1.0.0
Access Control	AC-17 (1)	Monitoring and Control	Remote debugging should be turned off for API Apps	1.0.0
Access Control	AC-17 (1)	Monitoring and Control	Remote debugging should be turned off for Function Apps	1.0.0
Access Control	AC-17 (1)	Monitoring and Control	Remote debugging should be turned off for Web Applications	1.0.0
Audit and Accountability	AU-6 (4)	Central Review and Analysis	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-6 (5)	Integrated Analysis of Audit Records	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-12	Audit Record Generation	Diagnostic logs in App Services should be enabled	2.0.0
Audit and Accountability	AU-12 (1)	System-wide and Time-correlated Audit Trail	Diagnostic logs in App Services should be enabled	2.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your API App	1.0.0
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Function Apps	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Configuration Management	CM-6	Configuration Settings	CORS should not allow every resource to access your Web Applications	1.0.0
Configuration Management	CM-6	Configuration Settings	Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Configuration Management	CM-6	Configuration Settings	Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	1.0.0
Configuration Management	CM-6	Configuration Settings	Function apps should have 'Client Certificates (Incoming client certificates)' enabled	1.0.1
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for API Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Function Apps	1.0.0
Configuration Management	CM-6	Configuration Settings	Remote debugging should be turned off for Web Applications	1.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Function App	2.0.0
Identification and Authentication	IA-2	Identification and Authentication (organizational Users)	Managed identity should be used in your Web App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your API App	2.0.0
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Function App	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Identification and Authentication	IA-4	Identifier Management	Managed identity should be used in your Web App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS only should be required in your API App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS only should be required in your Function App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	FTPS should be required in your Web App	2.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC-8	Transmission Confidentiality and Integrity	Web Application should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	API App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	FTPS only should be required in your API App	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	FTPS only should be required in your Function App	2.0.0

Domain	Control ID	Control Title	Policy	Policy Version
System and Communications Protection	SC-8 (1)	Cryptographic Protection	FTPS should be required in your Web App	2.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Function App should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Latest TLS version should be used in your API App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Latest TLS version should be used in your Function App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Latest TLS version should be used in your Web App	1.0.0
System and Communications Protection	SC-8 (1)	Cryptographic Protection	Web Application should only be accessible over HTTPS	1.0.0
System and Communications Protection	SC-28	Protection of Information at Rest	App Service Environment should enable internal encryption	1.0.0
System and Communications Protection	SC-28 (1)	Cryptographic Protection	App Service Environment should enable internal encryption	1.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI-2	Flaw Remediation	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'HTTP Version' is the latest, if used to run the API app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'HTTP Version' is the latest, if used to run the Function app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'HTTP Version' is the latest, if used to run the Web app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'Java version' is the latest, if used as a part of the API app	2.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'Java version' is the latest, if used as a part of the Function app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'Java version' is the latest, if used as a part of the Web app	2.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'PHP version' is the latest, if used as a part of the API app	2.1.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'PHP version' is the latest, if used as a part of the WEB app	2.1.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'Python version' is the latest, if used as a part of the API app	3.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'Python version' is the latest, if used as a part of the Function app	3.0.0
System and Information Integrity	SI-2 (6)	Removal of Previous Versions of Software and Firmware	Ensure that 'Python version' is the latest, if used as a part of the Web app	3.0.0

## UK OFFICIAL and UK NHS

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - UK OFFICIAL and UK NHS](#). For more information about this compliance standard, see [UK OFFICIAL](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Data in transit protection	1	Data in transit protection	API App should only be accessible over HTTPS	1.0.0
Data in transit protection	1	Data in transit protection	Function App should only be accessible over HTTPS	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Data in transit protection	1	Data in transit protection	Web Application should only be accessible over HTTPS	1.0.0
External interface protection	11	External interface protection	Remote debugging should be turned off for API Apps	1.0.0
External interface protection	11	External interface protection	Remote debugging should be turned off for Function Apps	1.0.0
External interface protection	11	External interface protection	Remote debugging should be turned off for Web Applications	1.0.0

## Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#).

# Security in Azure App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

This article shows you how [Azure App Service](#) helps secure your web app, mobile app back end, API app, and [function app](#). It also shows how you can further secure your app with the built-in App Service features.

The platform components of App Service, including Azure VMs, storage, network connections, web frameworks, management and integration features, are actively secured and hardened. App Service goes through vigorous compliance checks on a continuous basis to make sure that:

- Your app resources are [secured](#) from the other customers' Azure resources.
- [VM instances and runtime software are regularly updated](#) to address newly discovered vulnerabilities.
- Communication of secrets (such as connection strings) between your app and other Azure resources (such as [SQL Database](#)) stays within Azure and doesn't cross any network boundaries. Secrets are always encrypted when stored.
- All communication over the App Service connectivity features, such as [hybrid connection](#), is encrypted.
- Connections with remote management tools like Azure PowerShell, Azure CLI, Azure SDKs, REST APIs, are all encrypted.
- 24-hour threat management protects the infrastructure and platform against malware, distributed denial-of-service (DDoS), man-in-the-middle (MITM), and other threats.

For more information on infrastructure and platform security in Azure, see [Azure Trust Center](#).

The following sections show you how to further protect your App Service app from threats.

## HTTPS and Certificates

App Service lets you secure your apps with [HTTPS](#). When your app is created, its default domain name (<app\_name>.azurewebsites.net) is already accessible using HTTPS. If you [configure a custom domain for your app](#), you should also [secure it with a TLS/SSL certificate](#) so that client browsers can make secured HTTPS connections to your custom domain. There are several types of certificates supported by App Service:

- Free App Service Managed Certificate
- App Service certificate
- Third-party certificate
- Certificate imported from Azure Key Vault

For more information, see [Add a TLS/SSL certificate in Azure App Service](#).

## Insecure protocols (HTTP, TLS 1.0, FTP)

To secure your app against all unencrypted (HTTP) connections, App Service provides one-click configuration to enforce HTTPS. Unsecured requests are turned away before they even reach your application code. For more information, see [Enforce HTTPS](#).

[TLS](#) 1.0 is no longer considered secure by industry standards, such as [PCI DSS](#). App Service lets you disable outdated protocols by [enforcing TLS 1.1/1.2](#).

App Service supports both FTP and FTPS for deploying your files. However, FTPS should be used instead of FTP, if at all possible. When one or both of these protocols are not in use, you should [disable them](#).

## Static IP restrictions

By default, your App Service app accepts requests from all IP addresses from the internet, but you can limit that access to a small subset of IP addresses. App Service on Windows lets you define a list of IP addresses that are allowed to access your app. The allowed list can include individual IP addresses or a range of IP addresses defined by a subnet mask. For more information, see [Azure App Service Static IP Restrictions](#).

For App Service on Windows, you can also restrict IP addresses dynamically by configuring the *web.config*. For more information, see [Dynamic IP Security <dynamicIpSecurity>](#).

## Client authentication and authorization

Azure App Service provides turn-key authentication and authorization of users or client apps. When enabled, it can sign in users and client apps with little or no application code. You may implement your own authentication and authorization solution, or allow App Service to handle it for you instead. The authentication and authorization module handles web requests before handing them off to your application code, and it denies unauthorized requests before they reach your code.

App Service authentication and authorization support multiple authentication providers, including Azure Active Directory, Microsoft accounts, Facebook, Google, and Twitter. For more information, see [Authentication and authorization in Azure App Service](#).

## Service-to-service authentication

When authenticating against a back-end service, App Service provides two different mechanisms depending on your need:

- **Service identity** - Sign in to the remote resource using the identity of the app itself. App Service lets you easily create a [managed identity](#), which you can use to authenticate with other services, such as [Azure SQL Database](#) or [Azure Key Vault](#). For an end-to-end tutorial of this approach, see [Secure Azure SQL Database connection from App Service using a managed identity](#).
- **On-behalf-of (OBO)** - Make delegated access to remote resources on behalf of the user. With Azure Active Directory as the authentication provider, your App Service app can perform delegated sign-in to a remote service, such as [Microsoft Graph API](#) or a remote API app in App Service. For an end-to-end tutorial of this approach, see [Authenticate and authorize users end-to-end in Azure App Service](#).

## Connectivity to remote resources

There are three types of remote resources your app may need to access:

- [Azure resources](#)
- [Resources inside an Azure Virtual Network](#)
- [On-premises resources](#)

In each of these cases, App Service provides a way for you to make secure connections, but you should still observe security best practices. For example, always use encrypted connections even if the back-end resource allows unencrypted connections. Furthermore, make sure that your back-end Azure service allows the minimum set of IP addresses. You can find the outbound IP addresses for your app at [Inbound and outbound IP addresses in Azure App Service](#).

### Azure resources

When your app connects to Azure resources, such as [SQL Database](#) and [Azure Storage](#), the connection stays within Azure and doesn't cross any network boundaries. However, the connection goes through the shared networking in Azure, so always make sure that your connection is encrypted.

If your app is hosted in an [App Service environment](#), you should [connect to supported Azure services using Virtual Network service endpoints](#).

## Resources inside an Azure Virtual Network

Your app can access resources in an [Azure Virtual Network](#) through [Virtual Network integration](#). The integration is established with a Virtual Network using a point-to-site VPN. The app can then access the resources in the Virtual Network using their private IP addresses. The point-to-site connection, however, still traverses the shared networks in Azure.

To isolate your resource connectivity completely from the shared networks in Azure, create your app in an [App Service environment](#). Since an App Service environment is always deployed to a dedicated Virtual Network, connectivity between your app and resources within the Virtual Network is fully isolated. For other aspects of network security in an App Service environment, see [Network isolation](#).

### On-premises resources

You can securely access on-premises resources, such as databases, in three ways:

- [Hybrid connections](#) - Establishes a point-to-point connection to your remote resource through a TCP tunnel. The TCP tunnel is established using TLS 1.2 with shared access signature (SAS) keys.
- [Virtual Network integration](#) with site-to-site VPN - As described in [Resources inside an Azure Virtual Network](#), but the Virtual Network can be connected to your on-premises network through a [site-to-site VPN](#). In this network topology, your app can connect to on-premises resources like other resources in the Virtual Network.
- [App Service environment](#) with site-to-site VPN - As described in [Resources inside an Azure Virtual Network](#), but the Virtual Network can be connected to your on-premises network through a [site-to-site VPN](#). In this network topology, your app can connect to on-premises resources like other resources in the Virtual Network.

## Application secrets

Don't store application secrets, such as database credentials, API tokens, and private keys in your code or configuration files. The commonly accepted approach is to access them as [environment variables](#) using the standard pattern in your language of choice. In App Service, the way to define environment variables is through [app settings](#) (and, especially for .NET applications, [connection strings](#)). App settings and connection strings are stored encrypted in Azure, and they're decrypted only before being injected into your app's process memory when the app starts. The encryption keys are rotated regularly.

Alternatively, you can integrate your App Service app with [Azure Key Vault](#) for advanced secrets management. By [accessing the Key Vault with a managed identity](#), your App Service app can securely access the secrets you need.

## Network isolation

Except for the **Isolated** pricing tier, all tiers run your apps on the shared network infrastructure in App Service. For example, the public IP addresses and front-end load balancers are shared with other tenants. The **Isolated** tier gives you complete network isolation by running your apps inside a dedicated [App Service environment](#). An App Service environment runs in your own instance of [Azure Virtual Network](#). It lets you:

- Serve your apps through a dedicated public endpoint, with dedicated front ends.
- Serve internal application using an internal load balancer (ILB), which allows access only from inside your Azure Virtual Network. The ILB has an IP address from your private subnet, which provides total isolation of your apps from the internet.
- [Use an ILB behind a web application firewall \(WAF\)](#). The WAF offers enterprise-level protection to your public-facing applications, such as DDoS protection, URI filtering, and SQL injection prevention.

For more information, see [Introduction to Azure App Service Environments](#).

# App Service networking features

11/2/2021 • 19 minutes to read • [Edit Online](#)

You can deploy applications in Azure App Service in multiple ways. 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 several features in App Service to help you meet those needs. The challenge is knowing which feature to use to solve a given problem. This article will help you determine which feature to use, based on some example use cases.

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.
- The single-tenant App Service Environment (ASE) hosts Isolated SKU App Service plans directly in your Azure virtual network.

The features you use will depend on whether you're in the multitenant service or in an ASE.

## NOTE

Networking features are not available for [apps deployed in Azure Arc](#).

## Multitenant 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 multitenant 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 VNet Integration
Service endpoints	VNet Integration
Private endpoints	

Other than noted exceptions, you can use all of these features together. You can mix the features to solve your problems.

## Use cases and features

For any given use case, there might be a few ways to solve the problem. Choosing the best feature sometimes

goes beyond the use case itself. The following inbound use cases suggest how to use App Service networking features to solve problems with controlling traffic going 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
Restrict access to your app from resources in a virtual network	Service endpoints Internal Load Balancer (ILB) ASE Private endpoints
Expose your app on a private IP in your virtual network	ILB ASE Private endpoints Private IP for inbound traffic on an Application Gateway instance with service endpoints
Protect your app with a web application firewall (WAF)	Application Gateway and ILB ASE Application Gateway with private endpoints Application Gateway with service endpoints Azure Front Door with access restrictions
Load balance traffic to your apps in different regions	Azure Front Door with access restrictions
Load balance traffic in the same region	<a href="#">Application Gateway with service endpoints</a>

The following outbound use cases suggest how to use App Service networking features to solve outbound access needs for your app:

OUTBOUND USE CASE	FEATURE
Access resources in an Azure virtual network in the same region	VNet Integration ASE
Access resources in an Azure virtual network in a different region	VNet Integration and virtual network peering Gateway-required VNet Integration ASE and virtual network peering
Access resources secured with service endpoints	VNet Integration ASE
Access resources in a private network that's not connected to Azure	Hybrid Connections
Access resources across Azure ExpressRoute circuits	VNet Integration ASE
Secure outbound traffic from your web app	VNet Integration and network security groups ASE
Route outbound traffic from your web app	VNet Integration and route tables ASE

## 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.

Dashboard > vnet-integration-app - Properties

### vnet-integration-app - Properties

App Service

Search (Ctrl+ /)

Deployment Center

Status  
Running

URL  
[vnet-integration-app.azurewebsites.net](https://vnet-integration-app.azurewebsites.net)

Virtual IP address  
No IP-based SSL binding is configured

Mode  
Standard

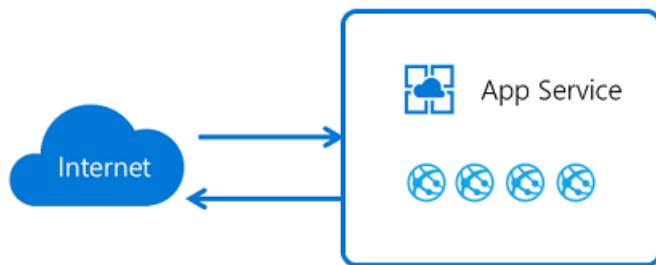
Outbound IP addresses  
13.90.143.69,13.90.251.106,13.90.192.153,13.90.199.220,40.71.205.9

Additional Outbound IP Addresses  
13.90.143.69,13.90.251.106,13.90.192.153,13.90.199.220,40.71.205.9,13.90.197.3,13.90.197.145

Deployment Trigger URL  
[https://\\$vnet-integration-app:cwQd7bev5y9020aPAptwZlsJjYqYMMMPZM5kZtFpgsfZ8svPq0ktyFpJif8S@vnet-integrati...](https://$vnet-integration-app:cwQd7bev5y9020aPAptwZlsJjYqYMMMPZM5kZtFpgsfZ8svPq0ktyFpJif8S@vnet-integrati...)

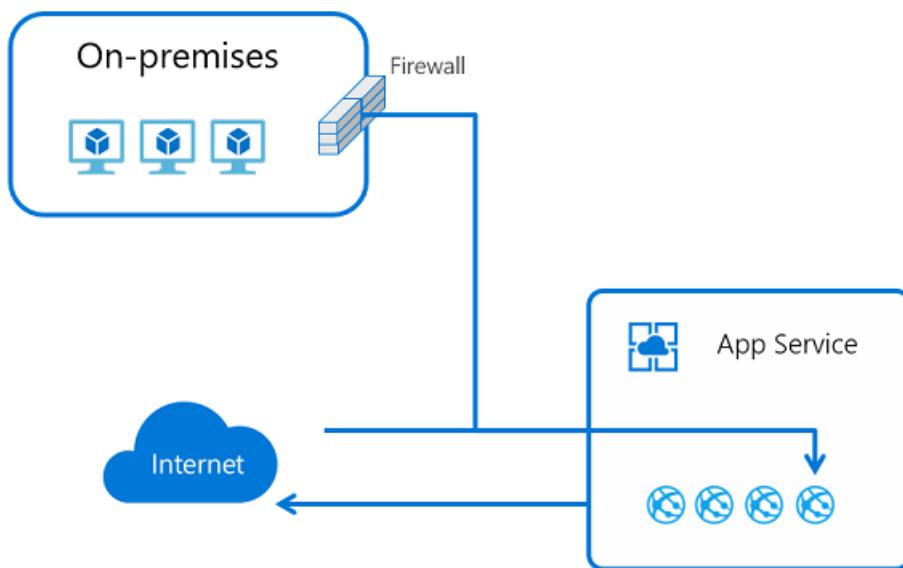
Properties

App Service has a number of endpoints that are used to manage the service. Those addresses are published in a separate document and are also in the `AppServiceManagement` IP service tag. The `AppServiceManagement` tag is used only in App Service Environments where you need to allow such traffic. The App Service inbound addresses are tracked in the `AppService` IP service tag. There's no IP service tag that contains the outbound addresses used by App Service.



### App-assigned address

The app-assigned address feature is an offshoot of the IP-based SSL capability. You access it by setting up SSL with your app. You can use this feature for IP-based SSL calls. You can also use it to give your app an address that only it has.



When you use an app-assigned address, your traffic still goes through the same front-end roles that handle all the incoming traffic into the App Service scale unit. But the address that's assigned to your app is used only by your app. Use cases for this feature:

- Support IP-based SSL needs for your app.
- Set a dedicated address for your app that's not shared.

To learn how to set an address on your app, see [Add a TLS/SSL certificate in Azure App Service](#).

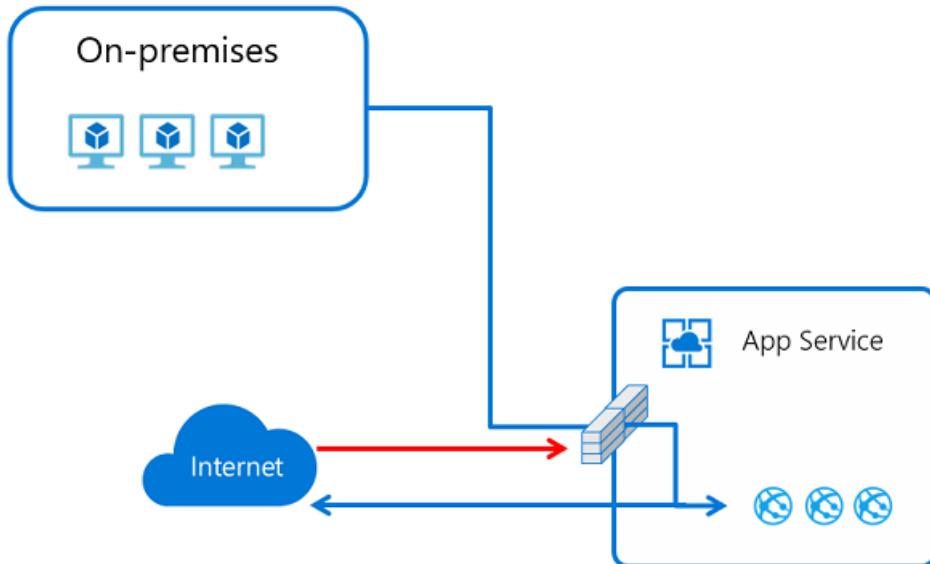
### Access restrictions

Access restrictions let you filter *inbound* requests. The filtering action takes place on the front-end roles that are upstream from the worker roles where your apps are running. Because the front-end roles are upstream from the workers, you can think of access restrictions as network-level protection for your apps.

This feature allows you to build a list of allow and deny rules that are evaluated in priority order. It's similar to the network security group (NSG) feature in Azure networking. You can use this feature in an ASE or in the multitenant service. When you use it with an ILB ASE, you can restrict access from private address blocks.

#### NOTE

Up to 512 access restriction rules can be configured per app.



#### IP-based access restriction rules

The IP-based access restrictions feature helps when you want to restrict the IP addresses that can be used to reach your app. Both IPv4 and IPv6 are supported. Some use cases for this feature:

- Restrict access to your app from a set of well-defined addresses.
- Restrict access to traffic coming through an external load-balancing service or other network appliances with known egress IP addresses.

To learn how to enable this feature, see [Configuring access restrictions](#).

#### NOTE

IP-based access restriction rules only handle virtual network address ranges when your app is in an App Service Environment. If your app is in the multitenant service, you need to use [service endpoints](#) to restrict traffic to select subnets in your virtual network.

#### Access restriction rules based on service endpoints

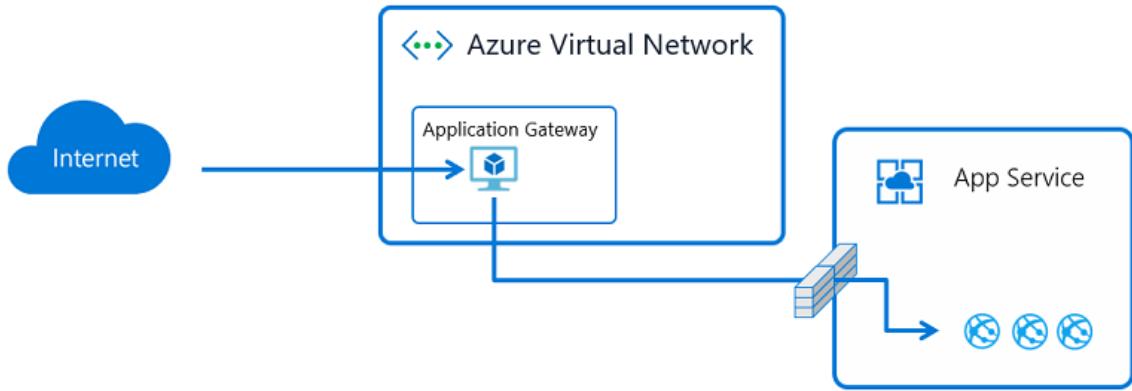
Service endpoints allow you to lock down *inbound* access to your app so that the source address must come from a set of subnets that you select. This feature works together with IP access restrictions. Service endpoints aren't compatible with remote debugging. If you want to use remote debugging with your app, your client can't be in a subnet that has service endpoints enabled. The process for setting service endpoints is similar to the process for setting IP access restrictions. You can build an allow/deny list of access rules that includes public addresses and subnets in your virtual networks.

#### NOTE

Access restriction rules based on service endpoints are not supported on apps that use IP-based SSL ([App-assigned address](#)).

Some use cases for this feature:

- Set up an application gateway with your app to lock down inbound traffic to your app.
- Restrict access to your app to resources in your virtual network. These resources can include VMs, ASEs, or even other apps that use VNet Integration.



To learn more about configuring service endpoints with your app, see [Azure App Service access restrictions](#).

#### **Access restriction rules based on service tags**

[Azure service tags](#) are well defined sets of IP addresses for Azure services. Service tags group the IP ranges used in various Azure services and is often also further scoped to specific regions. This allows you to filter *inbound* traffic from specific Azure services.

For a full list of tags and more information, visit the service tag link above. To learn how to enable this feature, see [Configuring access restrictions](#).

#### **Http header filtering for access restriction rules**

For each access restriction rule, you can add additional http header filtering. This allows you to further inspect the incoming request and filter based on specific http header values. Each header can have up to 8 values per rule. The following list of http headers is currently supported:

- X-Forwarded-For
- X-Forwarded-Host
- X-Azure-FDID
- X-FD-HealthProbe

Some use cases for http header filtering are:

- Restrict access to traffic from proxy servers forwarding the host name
- Restrict access to a specific Azure Front Door instance with a service tag rule and X-Azure-FDID header restriction

#### **Private endpoint**

Private endpoint is a network interface that connects you privately and securely to your Web App by Azure private link. Private endpoint uses a private IP address from your virtual network, effectively bringing the web app into your virtual network. This feature is only for *inbound* flows to your web app. For more information, see [Using private endpoints for Azure Web App](#).

Some use cases for this feature:

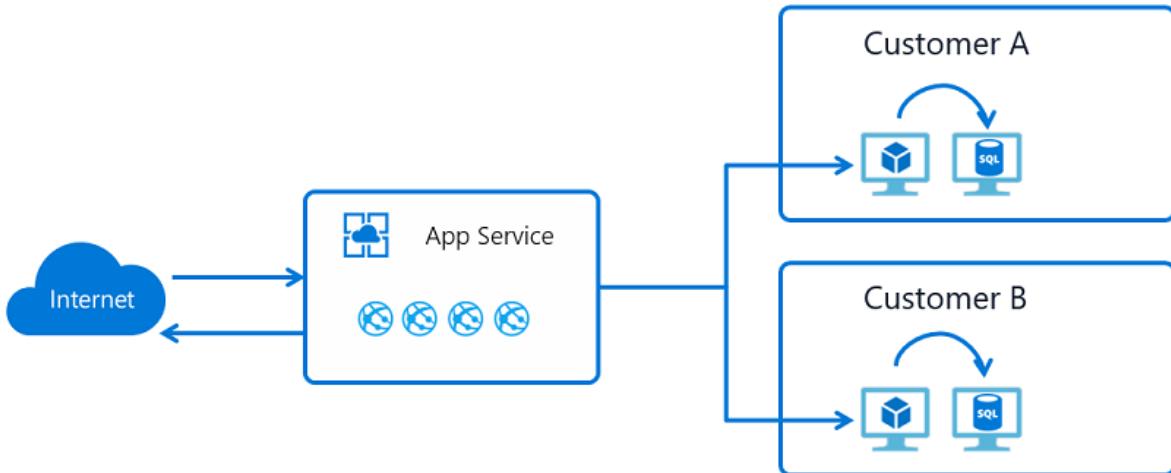
- Restrict access to your app from resources in a virtual network.
- Expose your app on a private IP in your virtual network.
- Protect your app with a WAF.

Private endpoints prevent data exfiltration because the only thing you can reach across the private endpoint is the app with which it's configured.

#### **Hybrid Connections**

App Service Hybrid Connections enables your apps to make *outbound* calls to specified TCP endpoints. The

endpoint can be on-premises, in a virtual network, or anywhere that allows outbound traffic to Azure on port 443. To use the feature, you need to install a relay agent called Hybrid Connection Manager on a Windows Server 2012 or newer host. Hybrid Connection Manager needs to be able to reach Azure Relay at port 443. You can download Hybrid Connection Manager from the App Service Hybrid Connections UI in the portal.



App Service Hybrid Connections is built on the Azure Relay Hybrid Connections capability. App Service uses a specialized form of the feature that only supports making outbound calls from your app to a TCP host and port. This host and port only need to resolve on the host where Hybrid Connection Manager is installed.

When the app, in App Service, does a DNS lookup on the host and port defined in your hybrid connection, the traffic automatically redirects to go through the hybrid connection and out of Hybrid Connection Manager. To learn more, see [App Service Hybrid Connections](#).

This feature is commonly used to:

- Access resources in private networks that aren't connected to Azure with a VPN or ExpressRoute.
- Support the migration of on-premises apps to App Service without the need to move supporting databases.
- Provide access with improved security to a single host and port per hybrid connection. Most networking features open access to a network. With Hybrid Connections, you can only reach the single host and port.
- Cover scenarios not covered by other outbound connectivity methods.
- Perform development in App Service in a way that allows the apps to easily use on-premises resources.

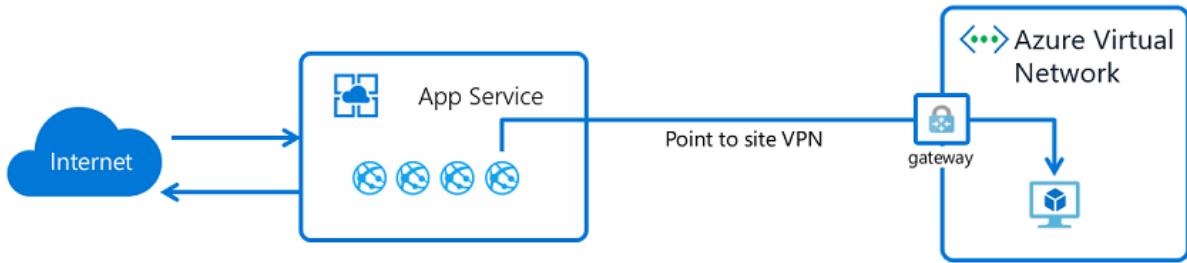
Because this feature enables access to on-premises resources without an inbound firewall hole, it's popular with developers. The other outbound App Service networking features are related to Azure Virtual Network. Hybrid Connections doesn't depend on going through a virtual network. It can be used for a wider variety of networking needs.

Note that App Service Hybrid Connections is unaware of what you're doing on top of it. So you can use it to access a database, a web service, or an arbitrary TCP socket on a mainframe. The feature essentially tunnels TCP packets.

Hybrid Connections is popular for development, but it's also used in production applications. It's great for accessing a web service or database, but it's not appropriate for situations that involve creating many connections.

### Gateway-required VNet Integration

Gateway-required App Service VNet Integration enables your app to make *outbound* requests into an Azure virtual network. The feature works by connecting the host your app is running on to a Virtual Network gateway on your virtual network by using a point-to-site VPN. When you configure the feature, your app gets one of the point-to-site addresses assigned to each instance. This feature enables you to access resources in either classic or Azure Resource Manager virtual networks in any region.



This feature solves the problem of accessing resources in other virtual networks. It can even be used to connect through a virtual network to either other virtual networks or on-premises. It doesn't work with ExpressRoute-connected virtual networks, but it does work with site-to-site VPN-connected networks. It's usually inappropriate to use this feature from an app in an App Service Environment (ASE) because the ASE is already in your virtual network. Use cases for this feature:

- Access resources on private IPs in your Classic virtual networks.
- Access resources on-premises if there's a site-to-site VPN.
- Access resources in cross region VNets that are not peered to a VNet in the region.

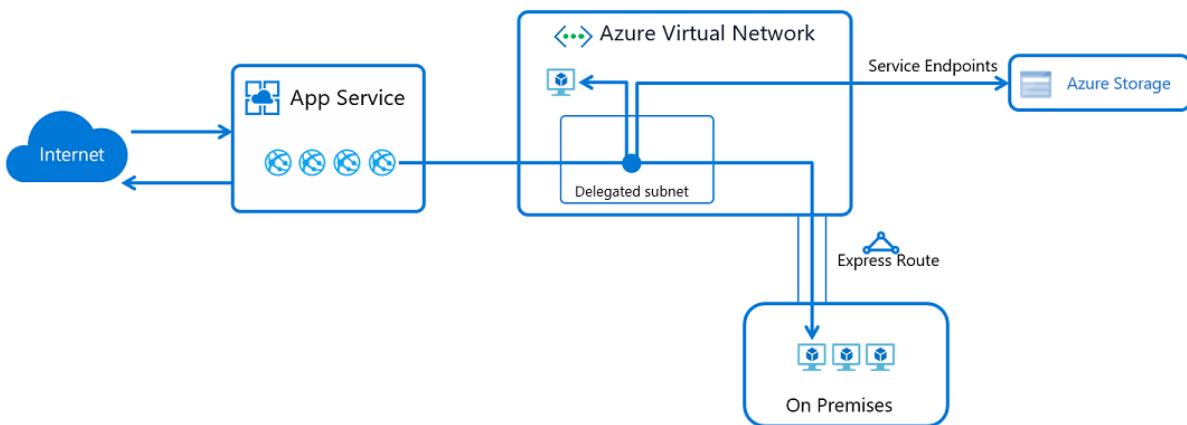
When this feature is enabled, your app will use the DNS server that the destination virtual network is configured with. For more information on this feature, see [App Service VNet Integration](#).

### Regional VNet Integration

Gateway-required VNet Integration is useful, but it doesn't solve the problem of accessing resources across ExpressRoute. On top of needing to reach across ExpressRoute connections, there's a need for apps to be able to make calls to services secured by service endpoint. Another VNet Integration capability can meet these needs.

The regional VNet Integration feature enables you to place the back end of your app in a subnet in a Resource Manager virtual network in the same region as your app. This feature isn't available from an App Service Environment, which is already in a virtual network. Use cases for this feature:

- Access resources in Resource Manager virtual networks in the same region.
- Access resources in peered virtual networks, including cross region connections.
- Access resources that are secured with service endpoints.
- Access resources that are accessible across ExpressRoute or VPN connections.
- Access resources in private networks without the need and cost of a Virtual Network gateway.
- Help to secure all outbound traffic.
- Force tunnel all outbound traffic.



To learn more, see [App Service VNet Integration](#).

### App Service Environment

An App Service Environment (ASE) is a single-tenant deployment of the Azure App Service that runs in your virtual network. Some cases such for this feature:

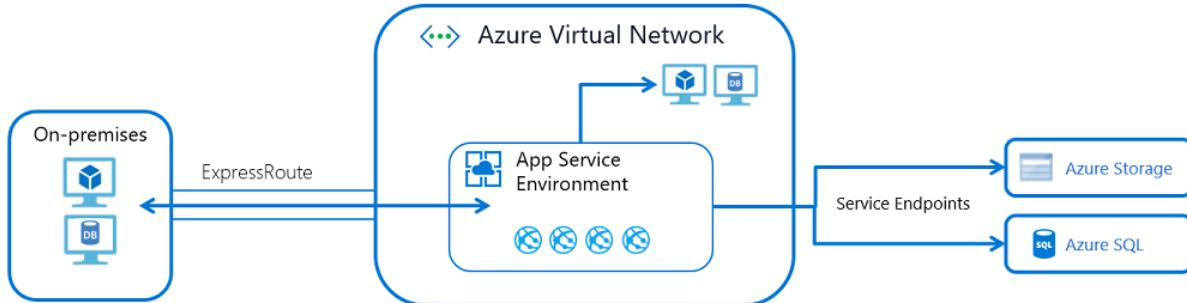
- Access resources in your virtual network.
- Access resources across ExpressRoute.
- Expose your apps with a private address in your virtual network.
- Access resources across service endpoints.
- Access resources across private endpoints.

With an ASE, you don't need to use VNet Integration because the ASE is already in your virtual network. If you want to access resources like SQL or Azure Storage over service endpoints, enable service endpoints on the ASE subnet. If you want to access resources in the virtual network or private endpoints in the virtual network, you don't need to do any additional configuration. If you want to access resources across ExpressRoute, you're already in the virtual network and don't need to configure anything on the ASE or the apps in it.

Because the apps in an ILB ASE can be exposed on a private IP address, you can easily add WAF devices to expose just the apps that you want to the internet and help keep the rest secure. This feature can help make the development of multitier applications easier.

Some things aren't currently possible from the multitenant service but are possible from an ASE. Here are some examples:

- Host your apps in a single-tenant service.
- Scale up to many more instances than are possible in the multitenant service.
- Load private CA client certificates for use by your apps with private CA-secured endpoints.
- Force TLS 1.1 across all apps hosted in the system without any ability to disable it at the app level.



The ASE provides the best story around isolated and dedicated app hosting, but it does involve some management challenges. Some things to consider before you use an operational ASE:

- An ASE runs inside your virtual network, but it does have dependencies outside the virtual network. Those dependencies must be allowed. For more information, see [Networking considerations for an App Service Environment](#).
- An ASE doesn't scale immediately like the multitenant service. You need to anticipate scaling needs rather than reactively scaling.
- An ASE does have a higher up-front cost. To get the most out of your ASE, you should plan to put many workloads into one ASE rather than using it for small efforts.
- The apps in an ASE can't selectively restrict access to some apps in the ASE and not others.
- An ASE is in a subnet, and any networking rules apply to all the traffic to and from that ASE. If you want to assign inbound traffic rules for just one app, use access restrictions.

## Combining features

The features noted for the multitenant service can be used together to solve more elaborate use cases. Two of

the more common use cases are described here, but they're just examples. By understanding what the various features do, you can meet nearly all your system architecture needs.

### Place an app into a virtual network

You might wonder how to put an app into a virtual network. If you put your app into a virtual network, the inbound and outbound endpoints for the app are within the virtual network. An ASE is the best way to solve this problem. But you can meet most of your needs within the multitenant service by combining features. For example, you can host intranet-only applications with private inbound and outbound addresses by:

- Creating an application gateway with private inbound and outbound addresses.
- Securing inbound traffic to your app with service endpoints.
- Using the new VNet Integration feature so the back end of your app is in your virtual network.

This deployment style won't give you a dedicated address for outbound traffic to the internet or the ability to lock down all outbound traffic from your app. It will give you a much of what you would only otherwise get with an ASE.

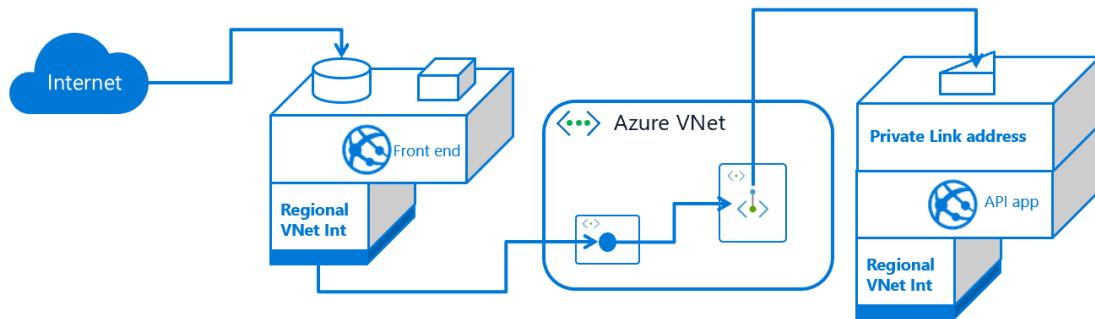
### Create multitier applications

A multitier application is an application in which the API back-end apps can be accessed only from the front-end tier. There are two ways to create a multitier application. Both start by using VNet Integration to connect your front-end web app to a subnet in a virtual network. Doing so will enable your web app to make calls into your virtual network. After your front-end app is connected to the virtual network, you need to decide how to lock down access to your API application. You can:

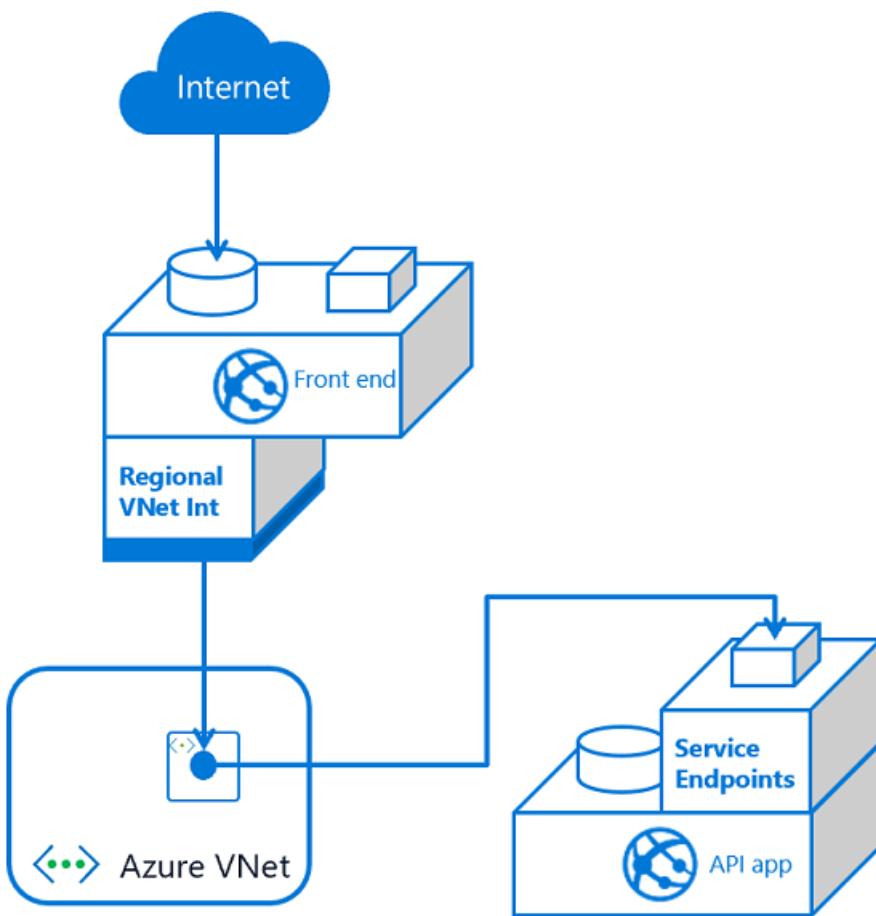
- Host both the front end and the API app in the same ILB ASE, and expose the front-end app to the internet by using an application gateway.
- Host the front end in the multitenant service and the back end in an ILB ASE.
- Host both the front end and the API app in the multitenant service.

If you're hosting both the front end and API app for a multitier application, you can:

- Expose your API application by using private endpoints in your virtual network:



- Use service endpoints to ensure inbound traffic to your API app comes only from the subnet used by your front-end web app:



Here are some considerations to help you decide which method to use:

- When you use service endpoints, you only need to secure traffic to your API app to the integration subnet. This helps to secure the API app, but you could still have data exfiltration from your front-end app to other apps in the app service.
- When you use private endpoints, you have two subnets at play, which adds complexity. Also, the private endpoint is a top-level resource and adds management overhead. The benefit of using private endpoints is that you don't have the possibility of data exfiltration.

Either method will work with multiple front ends. On a small scale, service endpoints are easier to use because you simply enable service endpoints for the API app on the front-end integration subnet. As you add more front-end apps, you need to adjust every API app to include service endpoints with the integration subnet. When you use private endpoints, there's more complexity, but you don't have to change anything on your API apps after you set a private endpoint.

### Line-of-business applications

Line-of-business (LOB) applications are internal applications that aren't normally exposed for access from the internet. These applications are called from inside corporate networks where access can be strictly controlled. If you use an ILB ASE, it's easy to host your line-of-business applications. If you use the multitenant service, you can either use private endpoints or use service endpoints combined with an application gateway. There are two reasons to use an application gateway with service endpoints instead of using private endpoints:

- You need WAF protection on your LOB apps.
- You want to load balance to multiple instances of your LOB apps.

If neither of these needs apply, you're better off using private endpoints. With private endpoints available in App Service, you can expose your apps on private addresses in your virtual network. The private endpoint you place in your virtual network can be reached across ExpressRoute and VPN connections.

Configuring private endpoints will expose your apps on a private address, but you'll need to configure DNS to

reach that address from on-premises. To make this configuration work, you'll need to forward the Azure DNS private zone that contains your private endpoints to your on-premises DNS servers. Azure DNS private zones don't support zone forwarding, but you can support zone forwarding by using a DNS server for that purpose. The [DNS Forwarder](#) template makes it easier to forward your Azure DNS private zone to your on-premises DNS servers.

## App Service ports

If you scan App Service, you'll find several ports that are exposed for inbound connections. There's no way to block or control access to these ports in the multitenant service. Here's the list of exposed ports:

USE	PORt OR PORTS
HTTP/HTTPS	80, 443
Management	454, 455
FTP/FTPS	21, 990, 10001-10300
Visual Studio remote debugging	4020, 4022, 4024
Web Deploy service	8172
Infrastructure use	7654, 1221

# Inbound and outbound IP addresses in Azure App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

Azure App Service is a multi-tenant service, except for [App Service Environments](#). Apps that are not in an App Service environment (not in the [Isolated tier](#)) share network infrastructure with other apps. As a result, the inbound and outbound IP addresses of an app can be different, and can even change in certain situations.

[App Service Environments](#) use dedicated network infrastructures, so apps running in an App Service environment get static, dedicated IP addresses both for inbound and outbound connections.

## How IP addresses work in App Service

An App Service app runs in an App Service plan, and App Service plans are deployed into one of the deployment units in the Azure infrastructure (internally called a webspace). Each deployment unit is assigned a set of virtual IP addresses, which includes one public inbound IP address and a set of [outbound IP addresses](#). All App Service plans in the same deployment unit, and app instances that run in them, share the same set of virtual IP addresses. For an App Service Environment (an App Service plan in [Isolated tier](#)), the App Service plan is the deployment unit itself, so the virtual IP addresses are dedicated to it as a result.

Because you're not allowed to move an App Service plan between deployment units, the virtual IP addresses assigned to your app usually remain the same, but there are exceptions.

## When inbound IP changes

Regardless of the number of scaled-out instances, each app has a single inbound IP address. The inbound IP address may change when you perform one of the following actions:

- Delete an app and recreate it in a different resource group (deployment unit may change).
- Delete the last app in a resource group *and* region combination and recreate it (deployment unit may change).
- Delete an existing IP-based TLS/SSL binding, such as during certificate renewal (see [Renew certificate](#)).

## Find the inbound IP

Just run the following command in a local terminal:

```
nslookup <app-name>.azurewebsites.net
```

## Get a static inbound IP

Sometimes you might want a dedicated, static IP address for your app. To get a static inbound IP address, you need to [secure a custom domain](#). If you don't actually need TLS functionality to secure your app, you can even upload a self-signed certificate for this binding. In an IP-based TLS binding, the certificate is bound to the IP address itself, so App Service provisions a static IP address to make it happen.

## When outbound IPs change

Regardless of the number of scaled-out instances, each app has a set number of outbound IP addresses at any

given time. Any outbound connection from the App Service app, such as to a back-end database, uses one of the outbound IP addresses as the origin IP address. The IP address to use is selected randomly at runtime, so your back-end service must open its firewall to all the outbound IP addresses for your app.

The set of outbound IP addresses for your app changes when you perform one of the following actions:

- Delete an app and recreate it in a different resource group (deployment unit may change).
- Delete the last app in a resource group *and* region combination and recreate it (deployment unit may change).
- Scale your app between the lower tiers (**Basic**, **Standard**, and **Premium**), the **PremiumV2**, and the **PremiumV3** tier (IP addresses may be added to or subtracted from the set).

You can find the set of all possible outbound IP addresses your app can use, regardless of pricing tiers, by looking for the `possibleOutboundIpAddresses` property or in the **Additional Outbound IP Addresses** field in the **Properties** blade in the Azure portal. See [Find outbound IPs](#).

## 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. They are listed in the **Outbound IP Addresses** field.

You can find the same information by running the following command in the [Cloud Shell](#).

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

```
(Get-AzWebApp -ResourceGroup <group_name> -name <app_name>).OutboundIpAddresses
```

To find *all* possible outbound IP addresses for your app, regardless of pricing tiers, click **Properties** in your app's left-hand navigation. They are listed in the **Additional Outbound IP Addresses** field.

You can find the same information by running the following command in the [Cloud Shell](#).

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

```
(Get-AzWebApp -ResourceGroup <group_name> -name <app_name>).PossibleOutboundIpAddresses
```

## Get a static outbound IP

You can control the IP address of outbound traffic from your app by using regional VNet integration together with a virtual network NAT gateway to direct traffic through a static public IP address. [Regional VNet integration](#) is available on **Standard**, **Premium**, **PremiumV2** and **PremiumV3** App Service plans. To learn more about this setup, see [NAT gateway integration](#).

## Next steps

Learn how to restrict inbound traffic by source IP addresses.

[Static IP restrictions](#)

# Using Private Endpoints for Azure Web App

11/2/2021 • 6 minutes to read • [Edit Online](#)

## IMPORTANT

Private Endpoint is available for Windows and Linux Web App, containerized or not, hosted on these App Service Plans : **PremiumV2, PremiumV3, Functions Premium** (sometimes referred to as the Elastic Premium plan).

You can use Private Endpoint for your Azure Web App to allow clients located in your private network to securely access the app over Private Link. The Private Endpoint uses an IP address from your Azure VNet address space. Network traffic between a client on your private network and the Web App traverses over the VNet and a Private Link on the Microsoft backbone network, eliminating exposure from the public Internet.

Using Private Endpoint for your Web App enables you to:

- Secure your Web App by configuring the Private Endpoint, eliminating public exposure.
- Securely connect to Web App from on-premises networks that connect to the VNet using a VPN or ExpressRoute private peering.
- Avoid any data exfiltration from your VNet.

If you just need a secure connection between your VNet and your Web App, a Service Endpoint is the simplest solution. If you also need to reach the web app from on-premises through an Azure Gateway, a regionally peered VNet, or a globally peered VNet, Private Endpoint is the solution.

For more information, see [Service Endpoints](#).

## Conceptual overview

A Private Endpoint is a special network interface (NIC) for your Azure Web App in a Subnet in your Virtual Network (VNet). When you create a Private Endpoint for your Web App, it provides secure connectivity between clients on your private network and your Web App. The Private Endpoint is assigned an IP Address from the IP address range of your VNet. The connection between the Private Endpoint and the Web App uses a secure [Private Link](#). Private Endpoint is only used for incoming flows to your Web App. Outgoing flows will not use this Private Endpoint, but you can inject outgoing flows to your network in a different subnet through the [VNet integration feature](#).

Each slot of an app is configured separately. You can plug up to 100 Private Endpoints per slot. You cannot share a Private Endpoint between slots.

The Subnet where you plug the Private Endpoint can have other resources in it, you don't need a dedicated empty Subnet. You can also deploy the Private Endpoint in a different region than the Web App.

## NOTE

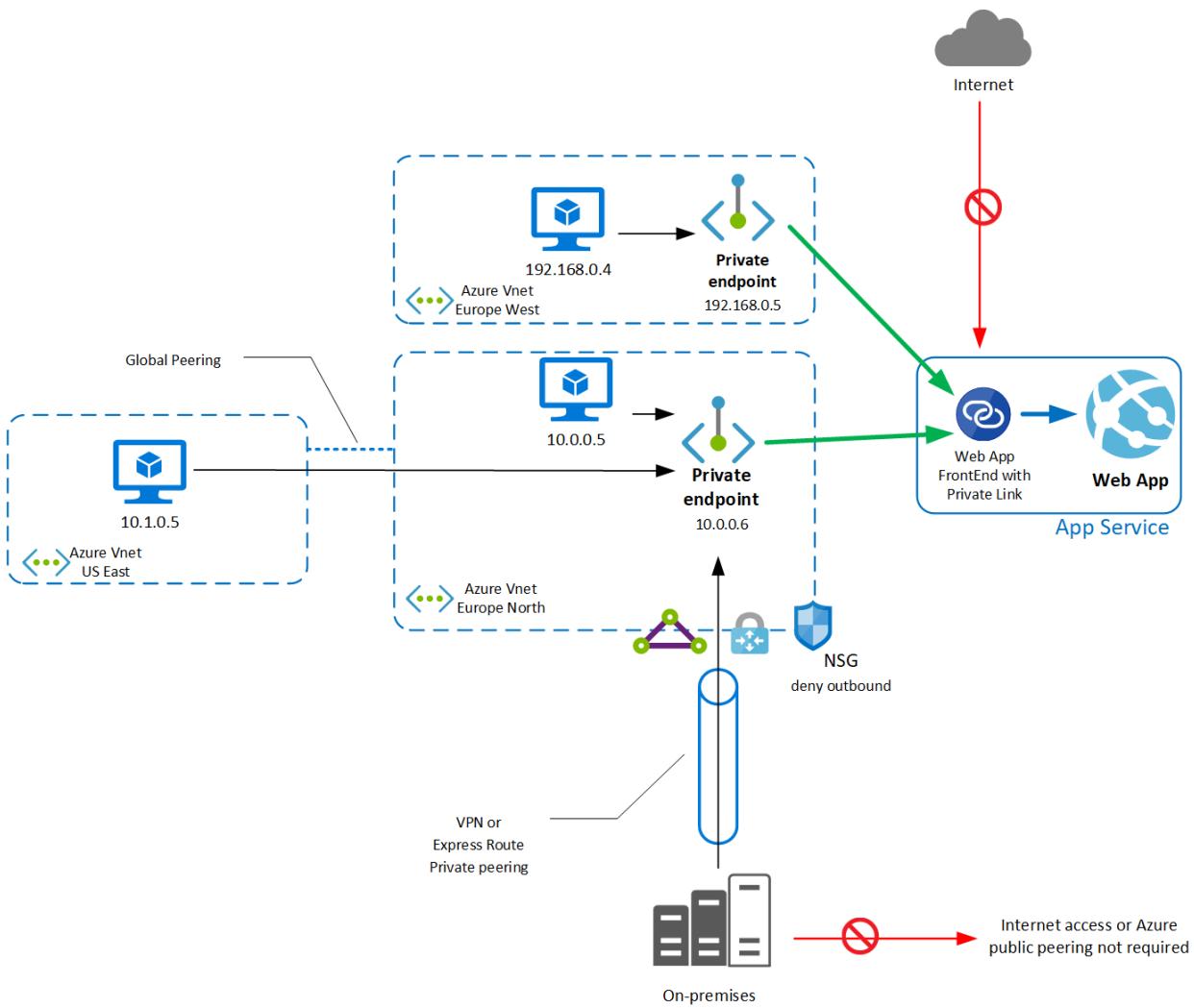
The VNet integration feature cannot use the same subnet as Private Endpoint, this is a limitation of the VNet integration feature.

From a security perspective:

- When you enable Private Endpoints to your Web App, you disable all public access.
- You can enable multiple Private Endpoints in others VNets and Subnets, including VNets in other regions.

- The IP address of the Private Endpoint NIC must be dynamic, but will remain the same until you delete the Private Endpoint.
- The NIC of the Private Endpoint cannot have an NSG associated.
- The Subnet that hosts the Private Endpoint can have an NSG associated, but you must disable the network policies enforcement for the Private Endpoint: see [Disable network policies for private endpoints](#). As a result, you cannot filter by any NSG the access to your Private Endpoint.
- When you enable Private Endpoint to your Web App, the [access restrictions](#) configuration of the Web App is not evaluated.
- You can eliminate the data exfiltration risk from the VNet by removing all NSG rules where destination is tag Internet or Azure services. When you deploy a Private Endpoint for a Web App, you can only reach this specific Web App through the Private Endpoint. If you have another Web App, you must deploy another dedicated Private Endpoint for this other Web App.

In the Web HTTP logs of your Web App, you will find the client source IP. This feature is implemented using the TCP Proxy protocol, forwarding the client IP property up to the Web App. For more information, see [Getting connection Information using TCP Proxy v2](#).



## DNS

When you use Private Endpoint for Web App, the requested URL must match the name of your Web App. By default `mywebappname.azurewebsites.net`.

By default, without Private Endpoint, the public name of your web app is a canonical name to the cluster. For example, the name resolution will be:

NAME	TYPE	VALUE
mywebapp.azurewebsites.net	CNAME	clustername.azurewebsites.windows.net
clustername.azurewebsites.windows.net	CNAME	cloudservicename.cloudapp.net
cloudservicename.cloudapp.net	A	40.122.110.154

When you deploy a Private Endpoint, we update the DNS entry to point to the canonical name mywebapp.privatelink.azurewebsites.net. For example, the name resolution will be:

NAME	TYPE	VALUE	REMARK
mywebapp.azurewebsites.net	CNAME	mywebapp.privatelink.azurewebsites.net	
mywebapp.privatelink.azurewebsites.net	CNAME	clustername.azurewebsites.windows.net	
clustername.azurewebsites.windows.net	CNAME	cloudservicename.cloudapp.net	
cloudservicename.cloudapp.net	A	40.122.110.154	<--This public IP is not your Private Endpoint, you will receive a 403 error

You must setup a private DNS server or an Azure DNS private zone, for tests you can modify the host entry of your test machine. The DNS zone that you need to create is: **privatelink.azurewebsites.net**. Register the record for your Web App with a A record and the Private Endpoint IP. For example, the name resolution will be:

NAME	TYPE	VALUE	REMARK
mywebapp.azurewebsites.net	CNAME	mywebapp.privatelink.azurewebsites.net	<--Azure creates this entry in Azure Public DNS to point the app service to the privatelink and this is managed by us
mywebapp.privatelink.azurewebsites.net	A	10.10.10.8	<--You manage this entry in your DNS system to point to your Private Endpoint IP address

After this DNS configuration you can reach your Web App privately with the default name mywebappname.azurewebsites.net. You must use this name, because the default certificate is issued for \*.azurewebsites.net.

If you need to use a custom DNS name, you must add the custom name in your Web App. The custom name must be validated like any custom name, using public DNS resolution. For more information, see [custom DNS validation](#).

For the Kudu console, or Kudu REST API (deployment with Azure DevOps self-hosted agents for example), you must create two records in your Azure DNS private zone or your custom DNS server.

NAME	TYPE	VALUE
mywebapp.privatelink.azurewebsites.net	A	PrivateEndpointIP
mywebapp.scm.privatelink.azurewebsites.net	A	PrivateEndpointIP

## Pricing

For pricing details, see [Azure Private Link pricing](#).

## Limitations

When you use Azure Function in Elastic Premium Plan with Private Endpoint, to run or execute the function in Azure Web portal, you must have direct network access or you will receive an HTTP 403 error. In other words, your browser must be able to reach the Private Endpoint to execute the function from the Azure Web portal.

You can connect up to 100 Private Endpoints to a particular Web App.

Remote Debugging functionality is not available when Private Endpoint is enabled for the Web App. The recommendation is to deploy the code to a slot and remote debug it there.

FTP access is provided through the inbound public IP address. Private Endpoint does not support FTP access to the Web App.

We are improving Private Link feature and Private Endpoint regularly, check [this article](#) for up-to-date information about limitations.

## Next steps

- To deploy Private Endpoint for your Web App through the portal, see [How to connect privately to a Web App with the Portal](#)
- To deploy Private Endpoint for your Web App using Azure CLI, see [How to connect privately to a Web App with Azure CLI](#)
- To deploy Private Endpoint for your Web App using PowerShell, see [How to connect privately to a Web App with PowerShell](#)
- To deploy Private Endpoint for your Web App using Azure template, see [How to connect privately to a Web App with Azure template](#)
- End-to-end example, how to connect a frontend web app to a secured backend web app with VNet injection and private endpoint with ARM template, see this [quickstart](#)
- End-to-end example, how to connect a frontend web app to a secured backend web app with VNet injection and private endpoint with terraform, see this [sample](#)

# Azure App Service Hybrid Connections

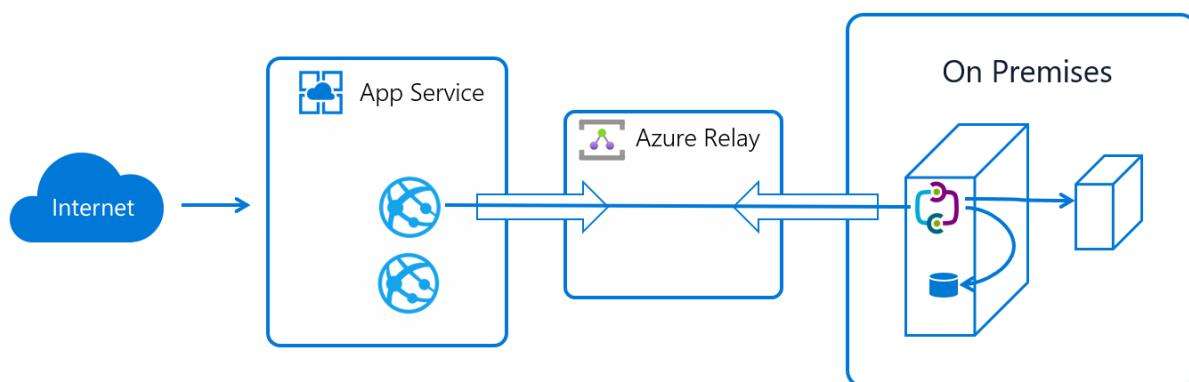
11/2/2021 • 11 minutes to read • [Edit Online](#)

Hybrid Connections is both a service in Azure and a feature in Azure App Service. As a service, it has uses and capabilities beyond those that are used in App Service. To learn more about Hybrid Connections and their usage outside App Service, see [Azure Relay Hybrid Connections](#).

Within App Service, Hybrid Connections can be used to access application resources in any network that can make outbound calls to Azure over port 443. Hybrid Connections provides access from your app to a TCP endpoint and doesn't enable a new way to access your app. As used in App Service, each Hybrid Connection correlates to a single TCP host and port combination. This enables your apps to access resources on any OS, provided it's a TCP endpoint. The Hybrid Connections feature doesn't know or care what the application protocol is, or what you are accessing. It simply provides network access.

## How it works

Hybrid Connections requires a relay agent to be deployed where it can reach both the desired endpoint as well as to Azure. The relay agent, Hybrid Connection Manager (HCM), calls out to Azure Relay over port 443. From the web app site, the App Service infrastructure also connects to Azure Relay on your application's behalf. Through the joined connections, your app is able to access the desired endpoint. The connection uses TLS 1.2 for security and shared access signature (SAS) keys for authentication and authorization.



When your app makes a DNS request that matches a configured Hybrid Connection endpoint, the outbound TCP traffic will be redirected through the Hybrid Connection.

### NOTE

This means that you should try to always use a DNS name for your Hybrid Connection. Some client software does not do a DNS lookup if the endpoint uses an IP address instead.

## App Service Hybrid Connection benefits

There are a number of benefits to the Hybrid Connections capability, including:

- Apps can access on-premises systems and services securely.
- The feature doesn't require an internet-accessible endpoint.
- It's quick and easy to set up. No gateways required.
- Each Hybrid Connection matches to a single host:port combination, helpful for security.
- It normally doesn't require firewall holes. The connections are all outbound over standard web ports.

- Because the feature is network level, it's agnostic to the language used by your app and the technology used by the endpoint.
- It can be used to provide access in multiple networks from a single app.
- It's supported in GA for Windows apps and Linux apps. It isn't supported for Windows container apps.

## Things you cannot do with Hybrid Connections

Things you cannot do with Hybrid Connections include:

- Mount a drive.
- Use UDP.
- Access TCP-based services that use dynamic ports, such as FTP Passive Mode or Extended Passive Mode.
- Support LDAP, because it can require UDP.
- Support Active Directory, because you cannot domain join an App Service worker.

## Add and Create Hybrid Connections in your app

To create a Hybrid Connection, go to the [Azure portal](#) and select your app. Select **Networking > Configure your Hybrid Connection endpoints**. Here you can see the Hybrid Connections that are configured for your app.

**Hybrid connections** hcportalapp

Refresh

**Hybrid connections**

App Service integration with hybrid connections enables your app to access a single TCP endpoint per hybrid connection. Here you can manage the new and classic hybrid connections used by your app. [Learn more](#)

App service plan (pricing tier): **hcportalplan (PremiumV3)**

Location: West Europe

Connections used: 0 / Connections quota 220

Download connection manager

**Add hybrid connection**

Name	Status	Endpoint	Namespace
No results			

To add a new Hybrid Connection, select **[+] Add hybrid connection**. You'll see a list of the Hybrid Connections that you already created. To add one or more of them to your app, select the ones you want, and then select **Add selected Hybrid Connection**.

**Add hybrid connection** hcportalapp

Create new hybrid connection Add selected hybrid connection Refresh

**Add hybrid connection**

To use a hybrid connection with your app select it from the list and click on 'Add selected hybrid connection'. Click on 'Create new hybrid connection' if you need to create a new one.

Name	Host	Port	Namespace	Location
myhybrid	localhost	80	mylistener	West Europe

If you want to create a new Hybrid Connection, select **Create new hybrid connection**. Specify the:

- Hybrid Connection name.
- Endpoint hostname.
- Endpoint port.
- Service Bus namespace you want to use.

## Create new hybrid connection ×

Hybrid connection Name \* ⓘ  
mysql-hybridconnection ✓

Endpoint Host \* ⓘ  
mysqldev ✓

Endpoint Port \* ⓘ  
3306 ✓

Servicebus namespace \* ⓘ  
 Create new  Select existing

Location \*  
West Europe ✓

Name \*  
hybridcon-sbns ✓

---

**OK**

Every Hybrid Connection is tied to a Service Bus namespace, and each Service Bus namespace is in an Azure region. It's important to try to use a Service Bus namespace in the same region as your app, to avoid network induced latency.

If you want to remove your Hybrid Connection from your app, right-click it and select **Disconnect**.

When a Hybrid Connection is added to your app, you can see details on it simply by selecting it.

Home > hcportalapp >

## Hybrid connections

hcportalapp

Refresh

### Hybrid connections

App Service integration with hybrid connections enables your app to access a single TCP endpoint per hybrid connection used by your app. [Learn more](#)

App service plan (pricing tier): **hcportalplan (PremiumV3)**

Location: West Europe

Connections used: 1 / 220

Name	Status	Endpoint
mysql-hybridconnection	Not connected	mysqldev : 3306

[Download connection manager](#)

[Add hybrid connection](#)

**GATEWAY CONNECTION STRING**  
Endpoint=sb://hybridc... [Edit](#)

[Disconnect](#)

**Properties**

hcportalapp

HYBRID CONNECTION NAME  
mysql-hybridconnection

ENDPOINT HOST  
mysqldev

ENDPOINT PORT  
3306

SERVICE BUS NAMESPACE  
hybridcon-sbns

NAMESPACE LOCATION  
West Europe

HYBRID CONNECTION MANAGERS  
0 connected

### Create a Hybrid Connection in the Azure Relay portal

In addition to the portal experience from within your app, you can create Hybrid Connections from within the Azure Relay portal. For a Hybrid Connection to be used by App Service, it must:

- Require client authorization.
- Have a metadata item, named endpoint, that contains a host:port combination as the value.

## Hybrid Connections and App Service plans

App Service Hybrid Connections are only available in Basic, Standard, Premium, and Isolated pricing SKUs. There are limits tied to the pricing plan.

PRICING PLAN	NUMBER OF HYBRID CONNECTIONS USABLE IN THE PLAN
Basic	5 per plan
Standard	25 per plan
Premium (v1-v3)	220 per app
Isolated (v1-v2)	220 per app

The App Service plan UI shows you how many Hybrid Connections are being used and by what apps.

The screenshot shows the Azure portal interface for managing hybrid connections. On the left, there's a sidebar with a 'Hybrid connections' section. The main area is titled 'Properties' for a hybrid connection named 'mysql-hybridconnection'. Key details shown include:

- HYBRID CONNECTION NAME:** mysql-hybridconnection
- ENDPOINT HOST:** mysqldev
- ENDPOINT PORT:** 3306
- SERVICE BUS NAMESPACE:** hybridcon-sbns
- NAMESPACE LOCATION:** West Europe
- CONNECTION MANAGER CONNECTIONS:** 0
- NUMBER OF CONNECTED SITES:** 1
- GATEWAY CONNECTION STRING:** Endpoint=sb://hybridcon-sbns.servicebus.windows.net/SharedAccessKeyName=defaultListener;Shared... (with a copy icon)

A table at the bottom lists one hybrid connection entry:

Name	Endpoint
mysql-hybridconnec...	3306 : mysqldev

Select the Hybrid Connection to see details. You can see all the information that you saw at the app view. You can also see how many other apps in the same plan are using that Hybrid Connection.

There's a limit on the number of Hybrid Connection endpoints that can be used in an App Service plan. Each Hybrid Connection used, however, can be used across any number of apps in that plan. For example, a single Hybrid Connection that is used in five separate apps in an App Service plan counts as one Hybrid Connection.

## Pricing

In addition to there being an App Service plan SKU requirement, there's an additional cost to using Hybrid Connections. There's a charge for each listener used by a Hybrid Connection. The listener is the Hybrid Connection Manager. If you had five Hybrid Connections supported by two Hybrid Connection Managers, that would be 10 listeners. For more information, see [Service Bus pricing](#).

## Hybrid Connection Manager

The Hybrid Connections feature requires a relay agent in the network that hosts your Hybrid Connection endpoint. That relay agent is called the Hybrid Connection Manager (HCM). To download HCM, from your app in the [Azure portal](#), select **Networking > Configure your Hybrid Connection endpoints**.

This tool runs on Windows Server 2012 and later. The HCM runs as a service and connects outbound to Azure Relay on port 443.

After installing HCM, you can run `HybridConnectionManagerUi.exe` to use the UI for the tool. This file is in the Hybrid Connection Manager installation directory. In Windows 10, you can also just search for *Hybrid Connection Manager UI* in your search box.

The screenshot shows the Hybrid Connection Manager (HCM) interface. At the top, there's a navigation bar with icons for Home, Connectors, and Help. Below it is a section titled "About" with a "Log Out" button. The main area features a table with columns: NAME, AZURE STATUS, SERVICE NAME, and ENDPOINT. A blue button labeled "Add a new Hybrid Connection" with a plus icon is located at the bottom left of the table. At the bottom right, there are two buttons: "Enter Manually" and "Refresh".

When you start the HCM UI, the first thing you see is a table that lists all the Hybrid Connections that are configured with this instance of the HCM. If you want to make any changes, first authenticate with Azure.

To add one or more Hybrid Connections to your HCM:

1. Start the HCM UI.
2. Select **Add a new Hybrid Connection**.

The screenshot shows the "Configure New Hybrid Connections" dialog box. It has a "Subscription" dropdown set to "Test Environment". Below it is a table with columns: NAME, REGION, SERVICE NAME, and ENDPOINT. Two entries are listed: "myhybrid" (westeurope, mylistener, localhost:80) and "mysql-hybridconnection" (westeurope, hybridcon-sbns, mysqldev:3306). At the bottom, there are "Save" and "Cancel" buttons, and a "Enter Manually" button. At the very bottom of the screen, there are "Enter Manually" and "Refresh" buttons.

3. Sign in with your Azure account to get your Hybrid Connections available with your subscriptions. The HCM doesn't continue to use your Azure account beyond that.
4. Choose a subscription.
5. Select the Hybrid Connections that you want the HCM to relay.

The screenshot shows the Hybrid Connection Manager (HCM) application window. At the top, there's a header bar with the title "Hybrid Connection Manager". Below the header, there's a section titled "About". The main content area is a table with four columns: "NAME", "AZURE STATUS", "SERVICE NAME", and "ENDPOINT". A row in the table contains the values: "mysql-hybridconnection", "Connected", "hybridcon-sbns", and "mysqldev:3306". Above this table, there's a blue button with a plus sign and the text "Add a new Hybrid Connection". At the bottom right of the main content area, there are two buttons: "Enter Manually" and "Refresh".

6. Select **Save**.

You can now see the Hybrid Connections you added. You can also select the configured Hybrid Connection to see details.

The screenshot shows the Hybrid Connection Manager (HCM) application window. At the top, there's a header bar with the title "Hybrid Connection Manager". Below the header, there's a section titled "About". The main content area is a table with four columns: "NAME", "AZURE STATUS", "SERVICE NAME", and "ENDPOINT". A row in the table is highlighted with a blue background, containing the values: "mysql-hybridconnection", "Connected", "hybridcon-chnc", and "mysqldev:3306". To the left of this table, there's a blue button with a plus sign and the text "Add a new Hybrid Connection". On the far left, there's a sidebar with the heading "Hybrid Connection Details" and a "Remove" link. Below this, there's a list of connection details: Name (mysql-hybridconnection), Namespace (hybridcon-sbns), Endpoint (mysqldev:3306), Status (Connected), Service Bus Endpoint (hybridcon-sbns.servicebus.windows.net), Azure IP Address (104.46.32.56), Azure Ports (80, 443), Created On (05-05-2021 10:51:28), and Last Updated (05-05-2021 10:51:38). At the bottom right of the main content area, there's a "Close" button.

To support the Hybrid Connections it's configured with, HCM requires:

- TCP access to Azure over port 443.
- TCP access to the Hybrid Connection endpoint.
- The ability to do DNS look-ups on the endpoint host and the Service Bus namespace.

#### NOTE

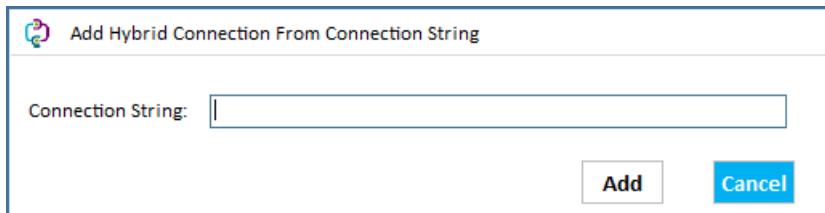
Azure Relay relies on Web Sockets for connectivity. This capability is only available on Windows Server 2012 or later. Because of that, HCM is not supported on anything earlier than Windows Server 2012.

### Redundancy

Each HCM can support multiple Hybrid Connections. Also, any given Hybrid Connection can be supported by multiple HCMs. The default behavior is to route traffic across the configured HCMs for any given endpoint. If you want high availability on your Hybrid Connections from your network, run multiple HCMs on separate machines. The load distribution algorithm used by the Relay service to distribute traffic to the HCMs is random assignment.

### Manually add a Hybrid Connection

To enable someone outside your subscription to host an HCM instance for a given Hybrid Connection, share the gateway connection string for the Hybrid Connection with them. You can see the gateway connection string in the Hybrid Connection properties in the [Azure portal](#). To use that string, select **Enter Manually** in the HCM, and paste in the gateway connection string.



### Upgrade

There are periodic updates to the Hybrid Connection Manager to fix issues or provide improvements. When upgrades are released, a popup will show up in the HCM UI. Applying the upgrade will apply the changes and restart the HCM.

## Adding a Hybrid Connection to your app programmatically

There's Azure CLI support for Hybrid Connections. The commands provided operate at both the app and the App Service plan level. The app level commands are:

```
az webapp hybrid-connection

Group
 az webapp hybrid-connection : Methods that list, add and remove hybrid-connections from webapps.
 This command group is in preview. It may be changed/removed in a future release.

Commands:
 add : Add a hybrid-connection to a webapp.
 list : List the hybrid-connections on a webapp.
 remove : Remove a hybrid-connection from a webapp.
```

The App Service plan commands enable you to set which key a given hybrid-connection will use. There are two keys set on each Hybrid Connection, a primary and a secondary. You can choose to use the primary or secondary key with the below commands. This enables you to switch keys for when you want to periodically regenerate your keys.

```
az appservice hybrid-connection --help

Group
 az appservice hybrid-connection : A method that sets the key a hybrid-connection uses.
 This command group is in preview. It may be changed/removed in a future release.
Commands:
 set-key : Set the key that all apps in an appservice plan use to connect to the hybrid-
 connections in that appservice plan.
```

## Secure your Hybrid Connections

An existing Hybrid Connection can be added to other App Service Web Apps by any user who has sufficient permissions on the underlying Azure Service Bus Relay. This means if you must prevent others from reusing that same Hybrid Connection (for example when the target resource is a service that doesn't have any additional security measures in place to prevent unauthorized access), you must lock down access to the Azure Service Bus Relay.

Anyone with `Reader` access to the Relay will be able to *see* the Hybrid Connection when attempting to add it to their Web App in the Azure portal, but they will not be able to *add* it as they lack the permissions to retrieve the connection string that is used to establish the relay connection. In order to successfully add the Hybrid Connection, they must have the `listKeys` permission (

`Microsoft.Relay/namespaces/hybridConnections/authorizationRules/listKeys/action`). The `Contributor` role or any other role that includes this permission on the Relay will allow users to use the Hybrid Connection and add it to their own Web Apps.

## Manage your Hybrid Connections

If you need to change the endpoint host or port for a Hybrid Connection, follow the steps below:

1. Remove the Hybrid Connection from the Hybrid Connection Manager on the local machine by selecting the connection and selecting **Remove** at the top left of the Hybrid Connection Details window.
2. Disconnect the Hybrid Connection from your App Service by navigating to **Hybrid Connections** in the App Service **Networking** page.
3. Navigate to the Relay for the endpoint you need to update and select **Hybrid Connections** under **Entities** in the left-hand navigation menu.
4. Select the Hybrid Connection you want to update and select **Properties** under **Settings** in the left-hand navigation menu.
5. Make your changes and hit **Save changes** at the top.
6. Return to the **Hybrid Connections** settings for your App Service and add the Hybrid Connection again. Ensure the endpoint is updated as intended. If you don't see the Hybrid Connection in the list, refresh in 5-10 minutes.
7. Return to the Hybrid Connection Manager on the local machine and add the connection again.

## Troubleshooting

The status of "Connected" means that at least one HCM is configured with that Hybrid Connection, and is able to reach Azure. If the status for your Hybrid Connection doesn't say **Connected**, your Hybrid Connection isn't configured on any HCM that has access to Azure. When your HCM shows **Not Connected** there are a few things to check:

- Does your host have outbound access to Azure on port 443? You can test from your HCM host using the PowerShell command `Test-NetConnection Destination -P Port`
- Is your HCM potentially in a bad state? Try restarting the 'Azure Hybrid Connection Manager Service'

local service.

- Do you have conflicting software installed? Hybrid Connection Manager cannot coexist with Biztalk Hybrid Connection Manager or Service Bus for Windows Server. Hence when installing HCM, any versions of these packages should be removed first.

If your status says **Connected** but your app cannot reach your endpoint then:

- make sure you're using a DNS name in your Hybrid Connection. If you use an IP address then the required client DNS lookup may not happen. If the client running in your web app doesn't do a DNS lookup, then the Hybrid Connection will not work
- check that the DNS name used in your Hybrid Connection can resolve from the HCM host. Check the resolution using *nslookup EndpointDNSname* where EndpointDNSname is an exact match to what is used in your Hybrid Connection definition.
- test access from your HCM host to your endpoint using the PowerShell command *Test-NetConnection EndpointDNSname -P Port* If you cannot reach the endpoint from your HCM host then check firewalls between the two hosts including any host-based firewalls on the destination host.
- if you're using App Service on Linux, make sure you're not using "localhost" as your endpoint host. Instead, use your machine name if you're trying to create a connection with a resource on your local machine.

In App Service, the **tcpping** command-line tool can be invoked from the Advanced Tools (Kudu) console. This tool can tell you if you have access to a TCP endpoint, but it doesn't tell you if you have access to a Hybrid Connection endpoint. When you use the tool in the console against a Hybrid Connection endpoint, you're only confirming that it uses a host:port combination.

If you have a command-line client for your endpoint, you can test connectivity from the app console. For example, you can test access to web server endpoints by using curl.

# Integrate your app with an Azure virtual network

11/2/2021 • 23 minutes to read • [Edit Online](#)

This article describes the Azure App Service VNet integration feature and how to set it up with apps in [Azure App Service](#). With [Azure virtual networks](#) (VNets), you can place many of your Azure resources in a non-internet-routable network. The VNet integration feature enables your apps to access resources in or through a VNet. VNet integration doesn't enable your apps to be accessed privately.

Azure App Service has two variations:

- The dedicated compute pricing tiers, which includes the Basic, Standard, Premium, PremiumV2, and PremiumV3.
- The App Service Environment which deploys directly into your VNet with dedicated supporting infrastructure and is using the Isolated and IsolatedV2 pricing tiers.

The VNet integration feature is used in App Service dedicated compute pricing tiers. If your app is in [App Service Environment](#), then it's already in a VNet and doesn't require use of the VNet integration feature to reach resources in the same VNet. For more information on all of the networking features, see [App Service networking features](#).

VNet integration gives your app access to resources in your VNet, but it doesn't grant inbound private access to your app from the VNet. Private site access refers to making an app accessible only from a private network, such as from within an Azure virtual network. VNet integration is used only to make outbound calls from your app into your VNet. The VNet integration feature behaves differently when it's used with VNet in the same region and with VNet in other regions. The VNet integration feature has two variations:

- **Regional VNet integration:** When you connect to virtual networks in the same region, you must have a dedicated subnet in the VNet you're integrating with.
- **Gateway-required VNet integration:** When you connect directly to VNet in other regions or to a classic virtual network in the same region, you need an Azure Virtual Network gateway created in the target VNet.

The VNet integration feature:

- Requires a **Standard, Premium, PremiumV2, PremiumV3, or Elastic Premium** App Service pricing tier.
- Supports TCP and UDP.
- Works with Azure App Service apps and function apps.

There are some things that VNet integration doesn't support, like:

- Mounting a drive.
- Windows Server Active Directory integration.
- NetBIOS.

Gateway-required VNet integration provides access to resources only in the target VNet or in networks connected to the target VNet with peering or VPNs. Gateway-required VNet integration doesn't enable access to resources available across Azure ExpressRoute connections or work with service endpoints.

Regardless of the version used, VNet integration gives your app access to resources in your VNet, but it doesn't grant inbound private access to your app from the VNet. Private site access refers to making your app accessible only from a private network, such as from within an Azure virtual network. VNet integration is only for making outbound calls from your app into your VNet.

[Learn how to enable VNet integration.](#)

# Regional VNet integration

Regional VNet integration supports connecting to a VNet in the same region and doesn't require a gateway.

Using regional VNet integration enables your app to access:

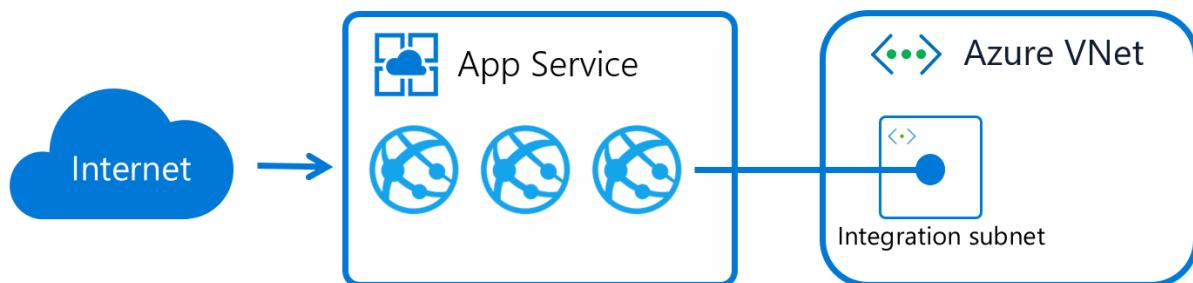
- Resources in the VNet you're integrated with.
- Resources in VNets peered to the VNet your app is integrated with including global peering connections.
- Resources across Azure ExpressRoute connections.
- Service endpoint secured services.
- Private endpoint enabled services.

When you use regional VNet integration, you can use the following Azure networking features:

- **Network security groups (NSGs)**: You can block outbound traffic with an NSG that's placed on your integration subnet. The inbound rules don't apply because you can't use VNet integration to provide inbound access to your app.
- **Route tables (UDRs)**: You can place a route table on the integration subnet to send outbound traffic where you want.

## How regional VNet integration works

Apps in App Service are hosted on worker roles. Regional VNet integration works by mounting virtual interfaces to the worker roles with addresses in the delegated subnet. Because the from address is in your VNet, it can access most things in or through your VNet like a VM in your VNet would. The networking implementation is different than running a VM in your VNet. That's why some networking features aren't yet available for this feature.



When regional VNet integration is enabled, your app makes outbound calls through your VNet. The outbound addresses that are listed in the app properties portal are the addresses still used by your app. However, if your outbound call is to a virtual machine or private endpoint in the integration VNet or peered VNet, the outbound address will be an address from the integration subnet. The private IP assigned to an instance is exposed via the environment variable, `WEBSITE_PRIVATE_IP`.

When all traffic routing is enabled, all outbound traffic is sent into your VNet. If all traffic routing is not enabled, only private traffic (RFC1918) and service endpoints configured on the integration subnet will be sent into the VNet and outbound traffic to the internet will go through the same channels as normal.

The feature supports only one virtual interface per worker. One virtual interface per worker means one regional VNet integration per App Service plan. All of the apps in the same App Service plan can use the same VNet integration. If you need an app to connect to an another VNet, you need to create another App Service plan. The virtual interface used isn't a resource that customers have direct access to.

Because of the nature of how this technology operates, the traffic that's used with VNet integration doesn't show up in Azure Network Watcher or NSG flow logs.

## Subnet requirements

VNet integration depends on a dedicated subnet. When you create a subnet, the Azure subnet loses five IPs from

the start. One address is used from the integration subnet for each plan instance. If you scale your app to four instances, then four addresses are used.

When you scale up or down in size, the required address space is doubled for a short period of time. This affects the real, available supported instances for a given subnet size. The following table shows both the maximum available addresses per CIDR block and the effect this has on horizontal scale:

CIDR BLOCK SIZE	MAX AVAILABLE ADDRESSES	MAX HORIZONTAL SCALE (INSTANCES)*
/28	11	5
/27	27	13
/26	59	29

\*Assumes that you'll need to scale up or down in either size or SKU at some point.

Since subnet size can't be changed after assignment, use a subnet that's large enough to accommodate whatever scale your app might reach. To avoid any issues with subnet capacity, you should use a /26 with 64 addresses.

When you want your apps in your plan to reach a VNet that's already connected to by apps in another plan, select a different subnet than the one being used by the pre-existing VNet integration.

## Routes

There are two types of routing to consider when configuring regional VNet integration. Application routing defines what traffic is routed from your application and into the VNet. Network routing is the ability to control how traffic is routed from your VNet and out.

### Application routing

When configuring application routing, you can either route all traffic or only private traffic (also known as [RFC1918](#) traffic) into your VNet. You configure this behavior through the Route All setting. If Route All is disabled, your app only routes private traffic into your VNet. If you want to route all of your outbound traffic into your VNet, make sure that Route All is enabled.

#### NOTE

- When Route All is enabled, all traffic is subject to the NSGs and UDRs that are applied to your integration subnet. When all traffic routing is enabled, outbound traffic is still sent from the addresses that are listed in your app properties, unless you provide routes that direct the traffic elsewhere.
- Windows containers do not support routing App Service Key Vault references or pulling custom container images over VNet integration.
- Regional VNet integration isn't able to use port 25.

Learn [how to configure application routing](#).

The Route All configuration setting is the recommended way of enabling routing of all traffic. Using the configuration setting will allow you to audit the behavior with [a built-in policy](#). The existing `WEBSITE_VNET_ROUTE_ALL` App Setting can still be used and you can enable all traffic routing with either setting.

### Network routing

You can use route tables to route outbound traffic from your app to wherever you want. Route tables affect your destination traffic. When Route All is disabled in [application routing](#), only private traffic (RFC1918) is affected by your route tables. Common destinations can include firewall devices or gateways. Routes that are set on your integration subnet won't affect replies to inbound app requests.

When you want to route all outbound traffic on-premises, you can use a route table to send all outbound traffic to your ExpressRoute gateway. If you do route traffic to a gateway, be sure to set routes in the external network to send any replies back.

Border Gateway Protocol (BGP) routes also affect your app traffic. If you have BGP routes from something like an ExpressRoute gateway, your app outbound traffic is affected. Similar to user defined routes, BGP routes affect traffic according to your routing scope setting.

## Network security groups

An app that uses regional VNet integration can use a [network security group](#) to block outbound traffic to resources in your VNet or the Internet. To block traffic to public addresses, you must ensure you enable [Route All](#) to the VNet. When Route All is not enabled, NSGs are only applied to RFC1918 traffic.

An NSG that's applied to your integration subnet is in effect regardless of any route tables applied to your integration subnet.

The inbound rules in an NSG do not apply to your app because VNet integration affects only outbound traffic from your app. To control inbound traffic to your app, use the Access Restrictions feature.

## Service endpoints

Regional VNet integration enables you to reach Azure services that are secured with service endpoints. To access a service endpoint-secured service, you must do the following steps:

- Configure regional VNet integration with your web app to connect to a specific subnet for integration.
- Go to the destination service and configure service endpoints against the integration subnet.

## Private endpoints

If you want to make calls to [private endpoints](#), then you must make sure that your DNS lookups resolve to the private endpoint. You can enforce this behavior in one of the following ways:

- Integrate with Azure DNS private zones. When your VNet doesn't have a custom DNS server, the integration is done automatically when the zones are linked to the VNet.
- Manage the private endpoint in the DNS server used by your app. To manage the configuration, you must know the private endpoint IP address and then point the endpoint you are trying to reach to that address using an A record.
- Configure your own DNS server to forward to Azure DNS private zones.

## Azure DNS private zones

After your app integrates with your VNet, it uses the same DNS server that your VNet is configured with, and if no custom DNS is specified it will use Azure default DNS and any private zones linked to the VNet.

## Limitations

There are some limitations with using regional VNet integration:

- The feature is available from all App Service scale units in Premium V2 and Premium V3. It's also available in Standard but only from newer App Service scale units. If you are on an older scale unit, you can only use the feature from a Premium V2 App Service plan. If you want to make sure you can use the feature in a Standard App Service plan, create your app in a Premium V3 App Service plan. Those plans are only supported on our newest scale units. You can scale down if you desire after the plan is created.
- The feature can't be used by Isolated plan apps that are in an App Service Environment.
- You can't reach resources across peering connections with classic virtual networks.
- The feature requires an unused subnet that's an IPv4 /28 block or larger in an Azure Resource Manager VNet.
- The app and the VNet must be in the same region.
- The integration VNet cannot have IPv6 address spaces defined.

- The integration subnet can be used by only one App Service plan.
- You can't delete a VNet with an integrated app. Remove the integration before you delete the VNet.
- You can have only one regional VNet integration per App Service plan. Multiple apps in the same App Service plan can use the same VNet.
- You can't change the subscription of an app or a plan while there's an app that's using regional VNet integration.

## Gateway-required VNet integration

Gateway-required VNet integration supports connecting to a VNet in another region or to a classic virtual network. Gateway-required VNet integration:

- Enables an app to connect to only one VNet at a time.
- Enables up to five VNets to be integrated within an App Service plan.
- Allows the same VNet to be used by multiple apps in an App Service plan without affecting the total number that can be used by an App Service plan. If you have six apps using the same VNet in the same App Service plan, that counts as one VNet being used.
- Supports a 99.9% SLA due to the SLA on the gateway.
- Enables your apps to use the DNS that the VNet is configured with.
- Requires a virtual network route-based gateway configured with an SSTP point-to-site VPN before it can be connected to an app.

You can't use gateway-required VNet integration:

- With a VNet connected with Azure ExpressRoute.
- From a Linux app.
- From a [Windows container](#).
- To access service endpoint secured resources.
- With a coexistence gateway that supports both ExpressRoute and point-to-site or site-to-site VPNs.

### Set up a gateway in your Azure virtual network

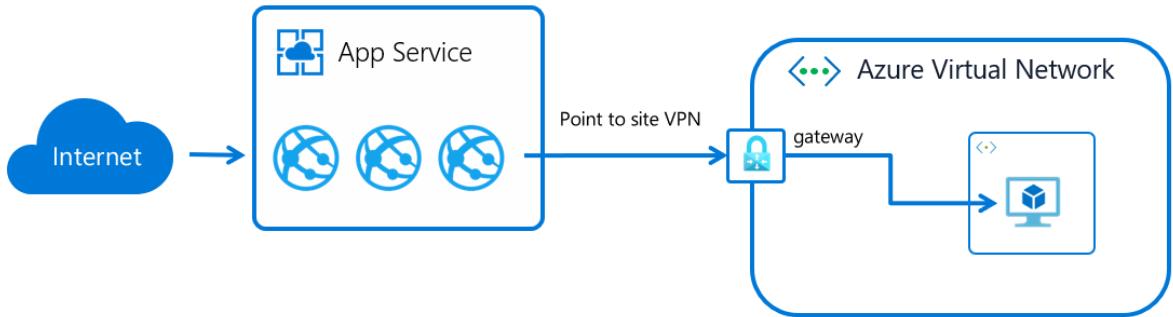
To create a gateway:

1. [Create the VPN gateway and subnet](#). Select a route-based VPN type.
2. [Set the point-to-site addresses](#). If the gateway isn't in the basic SKU, then IKEV2 must be disabled in the point-to-site configuration and SSTP must be selected. The point-to-site address space must be in the RFC 1918 address blocks 10.0.0.0/8, 172.16.0.0/12, and 192.168.0.0/16.

If you create the gateway for use with gateway-required VNet integration, you don't need to upload a certificate. Creating the gateway can take 30 minutes. You won't be able to integrate your app with your VNet until the gateway is created.

### How gateway-required VNet integration works

Gateway-required VNet integration is built on top of point-to-site VPN technology. Point-to-site VPNs limit network access to the virtual machine that hosts the app. Apps are restricted to send traffic out to the internet only through hybrid connections or through VNet integration. When your app is configured with the portal to use gateway-required VNet integration, a complex negotiation is managed on your behalf to create and assign certificates on the gateway and the application side. The result is that the workers used to host your apps are able to directly connect to the virtual network gateway in the selected VNet.



## Access on-premises resources

Apps can access on-premises resources by integrating with VNets that have site-to-site connections. If you use gateway-required VNet integration, update your on-premises VPN gateway routes with your point-to-site address blocks. When the site-to-site VPN is first set up, the scripts used to configure it should set up routes properly. If you add the point-to-site addresses after you create your site-to-site VPN, you need to update the routes manually. Details on how to do that vary per gateway and aren't described here. You can't have BGP configured with a site-to-site VPN connection.

No extra configuration is required for the regional VNet integration feature to reach through your VNet to on-premises resources. You simply need to connect your VNet to on-premises resources by using ExpressRoute or a site-to-site VPN.

### NOTE

The gateway-required VNet integration feature doesn't integrate an app with a VNet that has an ExpressRoute gateway. Even if the ExpressRoute gateway is configured in [coexistence mode](#), the VNet integration doesn't work. If you need to access resources through an ExpressRoute connection, use the regional VNet integration feature or an [App Service Environment](#), which runs in your VNet.

## Peering

If you use peering with the regional VNet integration, you don't need to do any additional configuration.

If you use gateway-required VNet integration with peering, you need to configure a few more items. To configure peering to work with your app:

1. Add a peering connection on the VNet your app connects to. When you add the peering connection, enable **Allow virtual network access** and select **Allow forwarded traffic** and **Allow gateway transit**.
2. Add a peering connection on the VNet that's being peered to the VNet you're connected to. When you add the peering connection on the destination VNet, enable **Allow virtual network access** and select **Allow forwarded traffic** and **Allow remote gateways**.
3. Go to the **App Service plan > Networking > VNet Integration** UI in the portal. Select the VNet your app connects to. Under the routing section, add the address range of the VNet that's peered with the VNet your app is connected to.

## Manage VNet integration

Connecting and disconnecting with a VNet is at an app level. Operations that can affect VNet integration across multiple apps are at the App Service plan level. From the app > **Networking > VNet Integration** portal, you can get details on your VNet. You can see similar information at the App Service plan level in the **App Service plan > Networking > VNet Integration** portal.

The only operation you can take in the app view of your VNet integration instance is to disconnect your app

from the VNet it's currently connected to. To disconnect your app from a VNet, select **Disconnect**. Your app is restarted when you disconnect from a VNet. Disconnecting doesn't change your VNet. The subnet or gateway isn't removed. If you then want to delete your VNet, first disconnect your app from the VNet and delete the resources in it, such as gateways.

The App Service plan VNet integration UI shows you all of the VNet integrations used by the apps in your App Service plan. To see details on each VNet, select the VNet you're interested in. There are two actions you can perform here for gateway-required VNet integration:

- **Sync network:** The sync network operation is used only for the gateway-required VNet integration feature. Performing a sync network operation ensures that your certificates and network information are in sync. If you add or change the DNS of your VNet, perform a sync network operation. This operation restarts any apps that use this VNet. This operation will not work if you are using an app and a vnet belonging to different subscriptions.
- **Add routes:** Adding routes drives outbound traffic into your VNet.

The private IP assigned to the instance is exposed via the environment variable, **WEBSITE\_PRIVATE\_IP**. Kudu console UI also shows the list of environment variables available to the Web App. This IP is assigned from the address range of the integrated subnet. For regional VNet integration, the value of **WEBSITE\_PRIVATE\_IP** is an IP from the address range of the delegated subnet, and for gateway-required VNet integration, the value is an IP from the address range of the Point-to-site address pool configured on the Virtual Network Gateway. This is the IP that will be used by the Web App to connect to the resources through the Azure virtual network.

#### NOTE

The value of **WEBSITE\_PRIVATE\_IP** is bound to change. However, it will be an IP within the address range of the integration subnet or the point-to-site address range, so you will need to allow access from the entire address range.

### Gateway-required VNet integration routing

The routes that are defined in your VNet are used to direct traffic into your VNet from your app. To send additional outbound traffic into the VNet, add those address blocks here. This capability only works with gateway-required VNet integration. Route tables don't affect your app traffic when you use gateway-required VNet integration the way that they do with regional VNet integration.

### Gateway-required VNet integration certificates

When gateway-required VNet integration is enabled, there's a required exchange of certificates to ensure the security of the connection. Along with the certificates are the DNS configuration, routes, and other similar things that describe the network.

If certificates or network information is changed, select **Sync Network**. When you select **Sync Network**, you cause a brief outage in connectivity between your app and your VNet. While your app isn't restarted, the loss of connectivity could cause your site to not function properly.

## Pricing details

The regional VNet integration feature has no extra charge for use beyond the App Service plan pricing tier charges.

Three charges are related to the use of the gateway-required VNet integration feature:

- **App Service plan pricing tier charges:** Your apps need to be in a Standard, Premium, PremiumV2, or PremiumV3 App Service plan. For more information on those costs, see [App Service pricing](#).
- **Data transfer costs:** There's a charge for data egress, even if the VNet is in the same datacenter. Those charges are described in [Data Transfer pricing details](#).

- **VPN gateway costs:** There's a cost to the virtual network gateway that's required for the point-to-site VPN. For more information, see [VPN gateway pricing](#).

## Troubleshooting

### NOTE

VNET integration is not supported for Docker Compose scenarios in App Service. Access Restrictions are ignored if their is a private endpoint present.

The feature is easy to set up, but that doesn't mean your experience will be problem free. If you encounter problems accessing your desired endpoint, there are some utilities you can use to test connectivity from the app console. There are two consoles that you can use. One is the Kudu console, and the other is the console in the Azure portal. To reach the Kudu console from your app, go to **Tools > Kudu**. You can also reach the Kudu console at [sitename].scm.azurewebsites.net. After the website loads, go to the **Debug console** tab. To get to the Azure portal-hosted console from your app, go to **Tools > Console**.

### Tools

In native Windows apps, the tools **ping**, **nslookup**, and **tracert** won't work through the console because of security constraints (they work in [custom Windows containers](#)). To fill the void, two separate tools are added. To test DNS functionality, we added a tool named **nameresolver.exe**. The syntax is:

```
nameresolver.exe hostname [optional: DNS Server]
```

You can use nameresolver to check the hostnames that your app depends on. This way you can test if you have anything misconfigured with your DNS or perhaps don't have access to your DNS server. You can see the DNS server that your app uses in the console by looking at the environmental variables WEBSITE\_DNS\_SERVER and WEBSITE\_DNS\_ALT\_SERVER.

### NOTE

nameresolver.exe currently doesn't work in custom Windows containers.

You can use the next tool to test for TCP connectivity to a host and port combination. This tool is called **tcpping** and the syntax is:

```
tcpping.exe hostname [optional: port]
```

The **tcpping** utility tells you if you can reach a specific host and port. It can show success only if there's an application listening at the host and port combination, and there's network access from your app to the specified host and port.

### Debug access to virtual network-hosted resources

A number of things can prevent your app from reaching a specific host and port. Most of the time it's one of these things:

- **A firewall is in the way.** If you have a firewall in the way, you hit the TCP timeout. The TCP timeout is 21 seconds in this case. Use the **tcpping** tool to test connectivity. TCP timeouts can be caused by many things beyond firewalls, but start there.
- **DNS isn't accessible.** The DNS timeout is 3 seconds per DNS server. If you have two DNS servers, the timeout is 6 seconds. Use nameresolver to see if DNS is working. You can't use nslookup, because that doesn't use the DNS your virtual network is configured with. If inaccessible, you could have a firewall or NSG

blocking access to DNS or it could be down.

If those items don't answer your problems, look first for things like:

### Regional VNet Integration

- Is your destination a non-RFC1918 address and you don't have Route All enabled?
- Is there an NSG blocking egress from your integration subnet?
- If you're going across Azure ExpressRoute or a VPN, is your on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your virtual network but not on-premises, check your routes.
- Do you have enough permissions to set delegation on the integration subnet? During regional VNet Integration configuration, your integration subnet is delegated to Microsoft.Web/serverFarms. The VNet Integration UI delegates the subnet to Microsoft.Web/serverFarms automatically. If your account doesn't have sufficient networking permissions to set delegation, you'll need someone who can set attributes on your integration subnet to delegate the subnet. To manually delegate the integration subnet, go to the Azure Virtual Network subnet UI and set the delegation for Microsoft.Web/serverFarms.

### Gateway-required VNet Integration

- Is the point-to-site address range in the RFC 1918 ranges (10.0.0.0-10.255.255.255 / 172.16.0.0-172.31.255.255 / 192.168.0.0-192.168.255.255)?
- Does the gateway show as being up in the portal? If your gateway is down, then bring it back up.
- Do certificates show as being in sync, or do you suspect that the network configuration was changed? If your certificates are out of sync or you suspect that a change was made to your virtual network configuration that wasn't synced with your ASPs, select **Sync Network**.
- If you're going across a VPN, is the on-premises gateway configured to route traffic back up to Azure? If you can reach endpoints in your virtual network but not on-premises, check your routes.
- Are you trying to use a coexistence gateway that supports both point to site and ExpressRoute? Coexistence gateways aren't supported with VNet Integration.

Debugging networking issues is a challenge because you can't see what's blocking access to a specific host:port combination. Some causes include:

- You have a firewall up on your host that prevents access to the application port from your point-to-site IP range. Crossing subnets often requires public access.
- Your target host is down.
- Your application is down.
- You had the wrong IP or hostname.
- Your application is listening on a different port than what you expected. You can match your process ID with the listening port by using "netstat -aon" on the endpoint host.
- Your network security groups are configured in such a manner that they prevent access to your application host and port from your point-to-site IP range.

You don't know what address your app actually uses. It could be any address in the integration subnet or point-to-site address range, so you need to allow access from the entire address range.

Additional debug steps include:

- Connect to a VM in your virtual network and attempt to reach your resource host:port from there. To test for TCP access, use the PowerShell command **test-netconnection**. The syntax is:

```
test-netconnection hostname [optional: -Port]
```

- Bring up an application on a VM and test access to that host and port from the console from your app by

using **tcpping**.

#### On-premises resources

If your app can't reach a resource on-premises, check if you can reach the resource from your virtual network. Use the **test-netconnection** PowerShell command to check for TCP access. If your VM can't reach your on-premises resource, your VPN or ExpressRoute connection might not be configured properly.

If your virtual network-hosted VM can reach your on-premises system but your app can't, the cause is likely one of the following reasons:

- Your routes aren't configured with your subnet or point-to-site address ranges in your on-premises gateway.
- Your network security groups are blocking access for your point-to-site IP range.
- Your on-premises firewalls are blocking traffic from your point-to-site IP range.
- You're trying to reach a non-RFC 1918 address by using the regional VNet Integration feature.

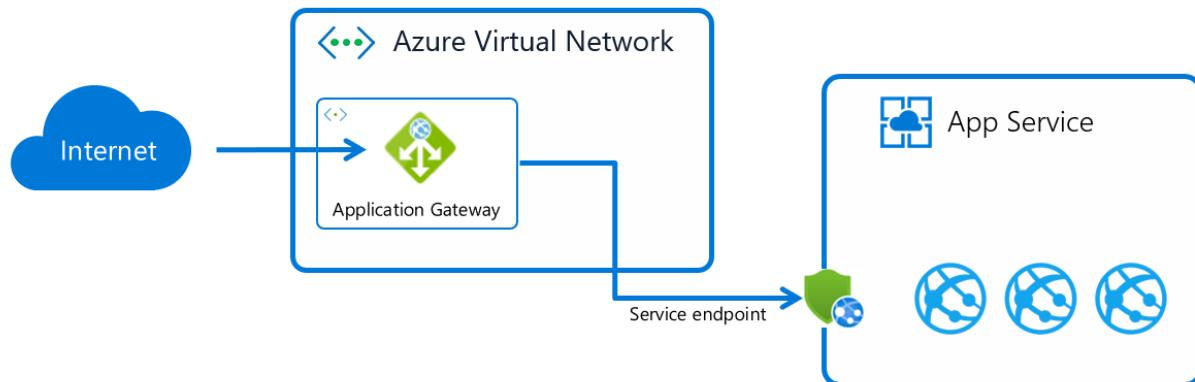
# Application Gateway integration

11/2/2021 • 4 minutes to read • [Edit Online](#)

There are three variations of App Service that require slightly different configuration of the integration with Azure Application Gateway. The variations include regular App Service - also known as multi-tenant, Internal Load Balancer (ILB) App Service Environment (ASE) and External ASE. This article will walk through how to configure it with App Service (multi-tenant) using service endpoint to secure traffic. The article will also discuss considerations around using private endpoint and integrating with ILB, and External ASE. Finally the article has considerations on scm/kudu site.

## Integration with App Service (multi-tenant)

App Service (multi-tenant) has a public internet facing endpoint. Using [service endpoints](#) you can allow traffic only from a specific subnet within an Azure Virtual Network and block everything else. In the following scenario, we'll use this functionality to ensure that an App Service instance can only receive traffic from a specific Application Gateway instance.



There are two parts to this configuration besides creating the App Service and the Application Gateway. The first part is enabling service endpoints in the subnet of the Virtual Network where the Application Gateway is deployed. Service endpoints will ensure all network traffic leaving the subnet towards the App Service will be tagged with the specific subnet ID. The second part is to set an access restriction of the specific web app to ensure that only traffic tagged with this specific subnet ID is allowed. You can configure it using different tools depending on preference.

## Using Azure portal

With Azure portal, you follow four steps to provision and configure the setup. If you have existing resources, you can skip the first steps.

1. Create an App Service using one of the Quickstarts in the App Service documentation, for example [.NET Core Quickstart](#)
2. Create an Application Gateway using the [portal Quickstart](#), but skip the Add backend targets section.
3. Configure [App Service as a backend in Application Gateway](#), but skip the Restrict access section.
4. Finally create the [access restriction using service endpoints](#).

You can now access the App Service through Application Gateway, but if you try to access the App Service directly, you should receive a 403 HTTP error indicating that the web site is stopped.

# Error 403 - Forbidden

The web app you have attempted to reach has blocked your access.

## Using Azure Resource Manager template

The [Resource Manager deployment template](#) will provision a complete scenario. The scenario consists of an App Service instance locked down with service endpoints and access restriction to only receive traffic from Application Gateway. The template includes many Smart Defaults and unique postfixes added to the resource names for it to be simple. To override them, you'll have to clone the repo or download the template and edit it.

To apply the template you can use the Deploy to Azure button found in the description of the template, or you can use appropriate PowerShell/CLI.

## Using Azure Command Line Interface

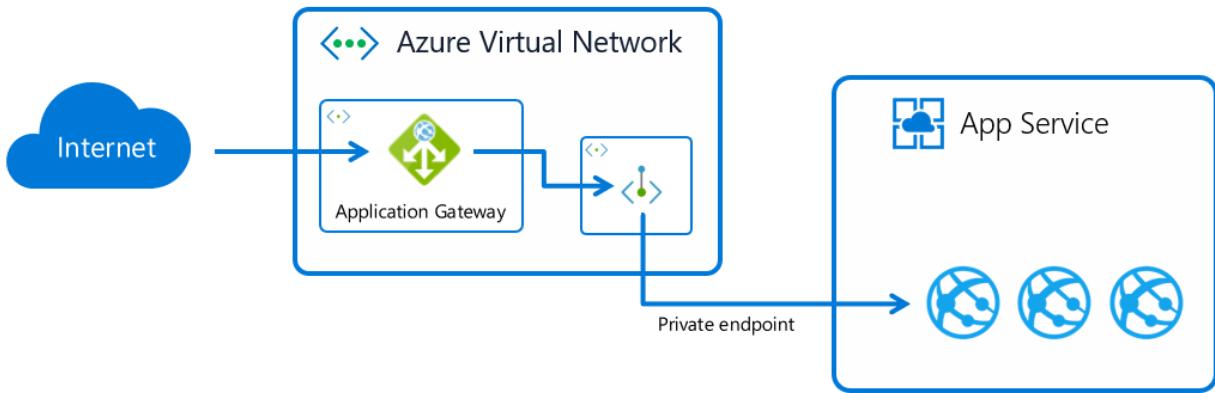
The [Azure CLI sample](#) will provision an App Service locked down with service endpoints and access restriction to only receive traffic from Application Gateway. If you only need to isolate traffic to an existing App Service from an existing Application Gateway, the following command is sufficient.

```
az webapp config access-restriction add --resource-group myRG --name myWebApp --rule-name AppGwSubnet --priority 200 --subnet mySubNetName --vnet-name myVnetName
```

In the default configuration, the command will ensure both setup of the service endpoint configuration in the subnet and the access restriction in the App Service.

## Considerations when using private endpoint

As an alternative to service endpoint, you can use private endpoint to secure traffic between Application Gateway and App Service (multi-tenant). You will need to ensure that Application Gateway can DNS resolve the private IP of the App Service apps or alternatively that you use the private IP in the backend pool and override the host name in the http settings.



## Considerations for ILB ASE

ILB ASE isn't exposed to the internet and traffic between the instance and an Application Gateway is therefore already isolated to the Virtual Network. The following [how-to guide](#) configures an ILB ASE and integrates it with an Application Gateway using Azure portal.

If you want to ensure that only traffic from the Application Gateway subnet is reaching the ASE, you can configure a Network security group (NSG) which affect all web apps in the ASE. For the NSG, you are able to specify the subnet IP range and optionally the ports (80/443). Make sure you don't override the [required NSG rules](#) for ASE to function correctly.

To isolate traffic to an individual web app you'll need to use ip-based access restrictions as service endpoints will not work for ASE. The IP address should be the private IP of the Application Gateway instance.

## Considerations for External ASE

External ASE has a public facing load balancer like multi-tenant App Service. Service endpoints don't work for ASE, and that's why you'll have to use ip-based access restrictions using the public IP of the Application Gateway instance. To create an External ASE using the Azure portal, you can follow this [Quickstart](#)

## Considerations for kudu/scm site

The scm site, also known as kudu, is an admin site, which exists for every web app. It isn't possible to reverse proxy the scm site and you most likely also want to lock it down to individual IP addresses or a specific subnet.

If you want to use the same access restrictions as the main site, you can inherit the settings using the following command.

```
az webapp config access-restriction set --resource-group myRG --name myWebApp --use-same-restrictions-for-scm-site
```

If you want to set individual access restrictions for the scm site, you can add access restrictions using the `--scm-site` flag like shown below.

```
az webapp config access-restriction add --resource-group myRG --name myWebApp --scm-site --rule-name KudoAccess --priority 200 --ip-address 208.130.0.0/16
```

## Next steps

For more information on the App Service Environment, see [App Service Environment documentation](#).

To further secure your web app, information about Web Application Firewall on Application Gateway can be

found in the [Azure Web Application Firewall documentation](#).

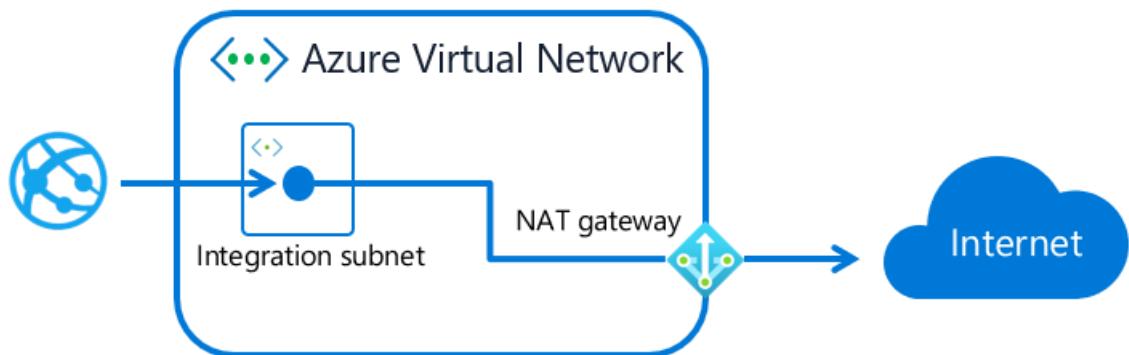
# Virtual Network NAT gateway integration

11/2/2021 • 2 minutes to read • [Edit Online](#)

NAT gateway is a fully managed, highly resilient service, which can be associated with one or more subnets and ensures that all outbound Internet-facing traffic will be routed through the gateway. With App Service, there are two important scenarios that you can use NAT gateway for.

The NAT gateway gives you a static predictable public IP for outbound Internet-facing traffic. It also significantly increases the available [SNAT ports](#) in scenarios where you have a high number of concurrent connections to the same public address/port combination.

For more information and pricing. Go to the [NAT gateway overview](#).



## NOTE

Using NAT gateway with App Service is dependent on regional VNet Integration, and therefore **Standard**, **Premium**, **PremiumV2** or **PremiumV3** App Service plan is required.

## Configuring NAT gateway integration

To configure NAT gateway integration with App Service, you need to complete the following steps:

- Configure regional VNet Integration with your app as described in [Integrate your app with an Azure virtual network](#)
- Ensure **Route All** is enabled for your VNet Integration so the Internet bound traffic will be affected by routes in your VNet.
- Provision a NAT gateway with a public IP and associate it with the VNet Integration subnet.

Set up NAT gateway through the portal:

- Go to the **Networking** UI in the App Service portal and select VNet Integration in the Outbound Traffic section. Ensure that your app is integrated with a subnet and **Route All** has been enabled.

[VNet Integration](#) ...  
vnet-integration-webapp  
[Disconnect](#) [Refresh](#) [Troubleshoot](#)

---

 [VNet Configuration](#)

Securely access resources available in or through your Azure VNet. [Learn more](#)

**Route All**  Enabled  
When Route All is enabled, all outbound traffic from the app will be affected by networking configurations. [Learn more](#)

**VNet Details**

VNet NAME	integration-vnet
LOCATION	West Europe

**VNet Address Space**

Start Address	10.42.0.0	End Address	10.42.255.255
---------------	-----------	-------------	---------------

**Subnet Details**

Subnet NAME	webapp-integration-subnet
-------------	---------------------------

2. On the Azure portal menu or from the **Home** page, select **Create a resource**. The **New** window appears.
3. Search for "NAT gateway" and select it from the list of results.
4. Fill in the **Basics** information and pick the region where your app is located.

[Home](#) > [Create a resource](#) > [NAT gateway](#) >

## Create network address translation (NAT) gateway ...

[Basics](#) [Outbound IP](#) [Subnet](#) [Tags](#) [Review + create](#)

Azure NAT gateway can be used to translate outbound flows from a virtual network to the public internet.  
[Learn more about NAT gateways](#).

### Project details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	<input type="text" value="Azure Subscription"/>
Resource group *	<input type="text" value="vnet-integration-setup"/> <a href="#">Create new</a>

### Instance details

NAT gateway name *	<input type="text" value="nat-gateway"/> 
Region *	<input type="text" value="(Europe) West Europe"/>
Availability zone ⓘ	<input type="text" value="None"/>
Idle timeout (minutes) * ⓘ	<input type="text" value="4"/> <small>4-120</small>

5. In the **Outbound IP** tab, create a new or select an existing public IP.

## Create network address translation (NAT) gateway

...

[Basics](#) [Outbound IP](#) [Subnet](#) [Tags](#) [Review + create](#)

Configure which public IP addresses and public IP prefixes to use. Each outbound IP address provides 64,000 SNAT ports for the NAT gateway resource to use. You can add up to 16 outbound IP addresses.

Note: While you do not have to complete this step to create a NAT gateway, the NAT gateway will not be functional and any subnet with this NAT gateway will not have outbound connectivity until you have added at least one public IP address or public IP prefix. You can also add and reconfigure which IP addresses are included after creating the NAT gateway.

The screenshot shows a modal dialog box titled "Add a public IP address". It contains the following fields:

- Name \***: nat-public-ip (highlighted with a green checkmark)
- SKU**: Basic (radio button)
- Assignment**: Dynamic (radio button)
- OK** and **Cancel** buttons

At the top left of the dialog, it says "0 selected". Below the dialog, there is a link "Create a new public IP address".

6. In the Subnet tab, select the subnet used for VNet Integration.

## Create network address translation (NAT) gateway

...

[Basics](#) [Outbound IP](#) [Subnet](#) [Tags](#) [Review + create](#)

To use the NAT gateway, at least one subnet must be selected. You can add and remove subnets after creating the NAT gateway.

Virtual network ⓘ

integration-vnet

▼

[Create new](#)

ⓘ Subnets that have any of the following resources are not shown because they are not compatible:

- A load balancer with a Basic SKU
- A public IP address with a Basic SKU
- An IPv6 address space
- An existing NAT gateway

 Subnet name

Subnet address range

 webapp-integration-subnet

10.42.0.0/24

[Manage subnets >](#)

7. Fill in tags if needed and **Create** the NAT gateway. After the NAT gateway is provisioned, click on the **Go to resource group** and select the new NAT gateway. You can see the public IP that your app will use for outbound Internet-facing traffic in the Outbound IP blade.

**Public IP addresses Change**

Name	IP address	DNS name
nat-public-ip	51.144.156.57	-

**Public IP prefixes Change**

No public IP prefixes

If you prefer using CLI to configure your environment, these are the important commands. As a prerequisite, you should create a Web App with VNet Integration configured.

Ensure Route All is configured for your VNet Integration (*Note: minimum az version required is 2.27*):

```
az webapp config set --resource-group [myResourceGroup] --name [myWebApp] --vnet-route-all-enabled
```

Create Public IP and NAT gateway:

```
az network public-ip create --resource-group [myResourceGroup] --name myPublicIP --sku standard --allocation static

az network nat gateway create --resource-group [myResourceGroup] --name myNATgateway --public-ip-addresses myPublicIP --idle-timeout 10
```

Associate the NAT gateway with the VNet Integration subnet:

```
az network vnet subnet update --resource-group [myResourceGroup] --vnet-name [myVnet] --name [myIntegrationSubnet] --nat-gateway myNATgateway
```

## Scaling NAT gateway

The same NAT gateway can be used across multiple subnets in the same Virtual Network allowing a NAT gateway to be used across multiple apps and App Service plans.

NAT gateway supports both public IP addresses and public IP prefixes. A NAT gateway can support up to 16 IP addresses across individual IP addresses and prefixes. Each IP address allocates 64,000 ports (SNAT ports) allowing up to 1M available ports. Learn more in the [Scaling section](#) of NAT gateway.

## Next steps

For more information on the NAT gateway, see [NAT gateway documentation](#).

For more information on VNet Integration, see [VNet Integration documentation](#).

# Controlling Azure App Service traffic with Azure Traffic Manager

11/2/2021 • 2 minutes to read • [Edit Online](#)

## NOTE

This article provides summary information for Microsoft Azure Traffic Manager as it relates to Azure App Service. More information about Azure Traffic Manager itself can be found by visiting the links at the end of this article.

## Introduction

You can use Azure Traffic Manager to control how requests from web clients are distributed to apps in Azure App Service. When App Service endpoints are added to an Azure Traffic Manager profile, Azure Traffic Manager keeps track of the status of your App Service apps (running, stopped, or deleted) so that it can decide which of those endpoints should receive traffic.

## Routing methods

Azure Traffic Manager uses four different routing methods. These methods are described in the following list as they pertain to Azure App Service.

- **Priority:** use a primary app for all traffic, and provide backups in case the primary or the backup apps are unavailable.
- **Weighted:** distribute traffic across a set of apps, either evenly or according to weights, which you define.
- **Performance:** when you have apps in different geographic locations, use the "closest" app in terms of the lowest network latency.
- **Geographic:** direct users to specific apps based on which geographic location their DNS query originates from.

For more information, see [Traffic Manager routing methods](#).

## App Service and Traffic Manager Profiles

To configure the control of App Service app traffic, you create a profile in Azure Traffic Manager that uses one of the four load balancing methods described previously, and then add the endpoints (in this case, App Service) for which you want to control traffic to the profile. Your app status (running, stopped, or deleted) is regularly communicated to the profile so that Azure Traffic Manager can direct traffic accordingly.

When using Azure Traffic Manager with Azure, keep in mind the following points:

- For app only deployments within the same region, App Service already provides failover and round-robin functionality without regard to app mode.
- For deployments in the same region that use App Service in conjunction with another Azure cloud service, you can combine both types of endpoints to enable hybrid scenarios.
- You can only specify one App Service endpoint per region in a profile. When you select an app as an endpoint for one region, the remaining apps in that region become unavailable for selection for that profile.
- The App Service endpoints that you specify in an Azure Traffic Manager profile appears under the **Domain Names** section on the Configure page for the app in the profile, but is not configurable there.

- After you add an app to a profile, the **Site URL** on the Dashboard of the app's portal page displays the custom domain URL of the app if you have set one up. Otherwise, it displays the Traffic Manager profile URL (for example, `contoso.trafficmanager.net`). Both the direct domain name of the app and the Traffic Manager URL are visible on the app's Configure page under the **Domain Names** section.
- Your custom domain names work as expected, but in addition to adding them to your apps, you must also configure your DNS map to point to the Traffic Manager URL. For information on how to set up a custom domain for an App Service app, see [Configure a custom domain name in Azure App Service with Traffic Manager integration](#).
- You can only add apps that are in standard or premium mode to an Azure Traffic Manager profile.
- Adding an app to a Traffic Manager profile causes the app to be restarted.

## Next Steps

For a conceptual and technical overview of Azure Traffic Manager, see [Traffic Manager Overview](#).

# Azure App Service Local Cache overview

11/2/2021 • 7 minutes to read • [Edit Online](#)

## NOTE

Local cache is not supported in function apps or containerized App Service apps, such as in [Windows Containers](#) or in [App Service on Linux](#). A version of local cache that is available for these app types is [App Cache](#).

Azure App Service content is stored on Azure Storage and is surfaced in a durable manner as a content share. This design is intended to work with a variety of apps and has the following attributes:

- The content is shared across multiple virtual machine (VM) instances of the app.
- The content is durable and can be modified by running apps.
- Log files and diagnostic data files are available under the same shared content folder.
- Publishing new content directly updates the content folder. You can immediately view the same content through the SCM website and the running app (typically some technologies such as ASP.NET do initiate an app restart on some file changes to get the latest content).

While many apps use one or all of these features, some apps just need a high-performance, read-only content store that they can run from with high availability. These apps can benefit from a VM instance of a specific local cache.

The Azure App Service Local Cache feature provides a web role view of your content. This content is a write-but-discard cache of your storage content that is created asynchronously on-site startup. When the cache is ready, the site is switched to run against the cached content. Apps that run on Local Cache have the following benefits:

- They are immune to latencies that occur when they access content on Azure Storage.
- They are immune to the planned upgrades or unplanned downtimes and any other disruptions with Azure Storage that occur on servers that serve the content share.
- They have fewer app restarts due to storage share changes.

## How the local cache changes the behavior of App Service

- *D:\home* points to the local cache, which is created on the VM instance when the app starts up. *D:\local* continues to point to the temporary VM-specific storage.
- The local cache contains a one-time copy of the */site* and */siteextensions* folders of the shared content store, at *D:\home\site* and *D:\home\siteextensions*, respectively. The files are copied to the local cache when the app starts. The size of the two folders for each app is limited to 1 GB by default, but can be increased to 2 GB. Note that as the cache size increases, it will take longer to load the cache. If you've increased local cache limit to 2 GB and the copied files exceed the maximum size of 2 GB, App Service silently ignores local cache and reads from the remote file share. If there is no limit defined or the limit is set to anything lower than 2 GB and the copied files exceeds the limit, the deployment or swap may fail with an error.
- The local cache is read-write. However, any modification is discarded when the app moves virtual machines or gets restarted. Do not use the local cache for apps that store mission-critical data in the content store.
- *D:\home\LogFiles* and *D:\home\Data* contain log files and app data. The two subfolders are stored locally on the VM instance, and are copied to the shared content store periodically. Apps can persist log files and data by writing them to these folders. However, the copy to the shared content store is best-effort, so it is possible for log files and data to be lost due to a sudden crash of a VM instance.
- [Log streaming](#) is affected by the best-effort copy. You could observe up to a one-minute delay in the

streamed logs.

- In the shared content store, there is a change in the folder structure of the *LogFiles* and *Data* folders for apps that use the local cache. There are now subfolders in them that follow the naming pattern of "unique identifier" + time stamp. Each of the subfolders corresponds to a VM instance where the app is running or has run.
- Other folders in *D:\home* remain in the local cache and are not copied to the shared content store.
- App deployment through any supported method publishes directly to the durable shared content store. To refresh the *D:\home\site* and *D:\home\siteextensions* folders in the local cache, the app needs to be restarted. To make the lifecycle seamless, see the information later in this article.
- The default content view of the SCM site continues to be that of the shared content store.

## Enable Local Cache in App Service

### NOTE

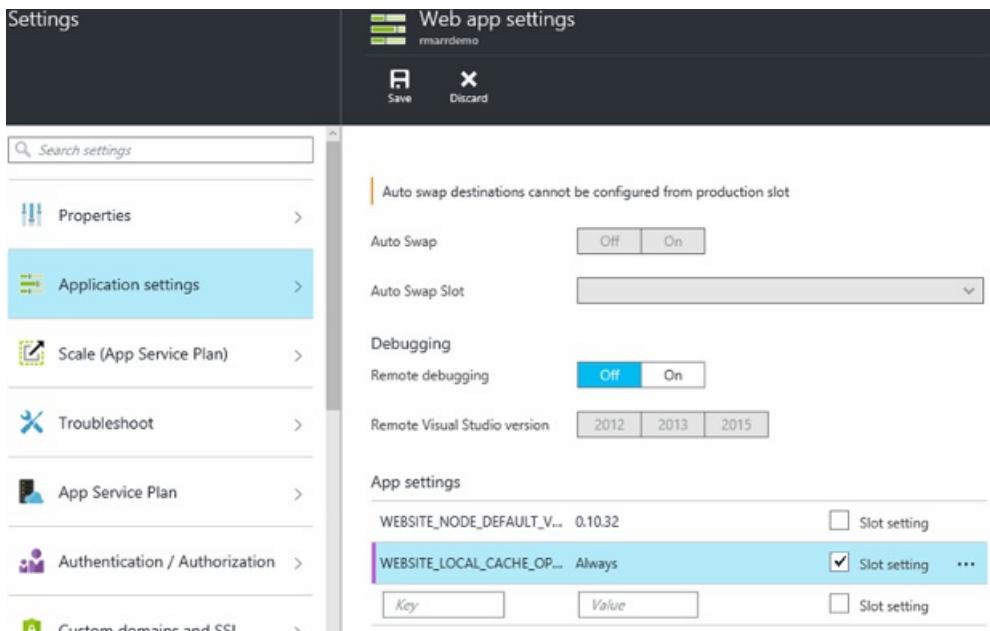
Local Cache is not supported in the F1 or D1 tier.

You configure Local Cache by using a combination of reserved app settings. You can configure these app settings by using the following methods:

- [Azure portal](#)
- [Azure Resource Manager](#)

### Configure Local Cache by using the Azure portal

You enable Local Cache on a per-web-app basis by using this app setting: `WEBSITE_LOCAL_CACHE_OPTION` = `Always`



### Configure Local Cache by using Azure Resource Manager

```

...
{
 "apiVersion": "2015-08-01",
 "type": "config",
 "name": "appsettings",
 "dependsOn": [
 "[resourceId('Microsoft.Web/sites/', variables('siteName'))]"
],
 "properties": {
 "WEBSITE_LOCAL_CACHE_OPTION": "Always",
 "WEBSITE_LOCAL_CACHE_SIZEINMB": "1000"
 }
}
...

```

## Change the size setting in Local Cache

By default, the local cache size is 1 GB. This includes the /site and /siteextensions folders that are copied from the content store, as well as any locally created logs and data folders. To increase this limit, use the app setting `WEBSITE_LOCAL_CACHE_SIZEINMB`. You can increase the size up to 2 GB (2000 MB) per app. Note that it will take longer to load local cache as the size increases.

## Best practices for using App Service Local Cache

We recommend that you use Local Cache in conjunction with the [Staging Environments](#) feature.

- Add the *sticky* app setting `WEBSITE_LOCAL_CACHE_OPTION` with the value `Always` to your **Production** slot. If you're using `WEBSITE_LOCAL_CACHE_SIZEINMB`, also add it as a sticky setting to your Production slot.
- Create a **Staging** slot and publish to your Staging slot. You typically don't set the staging slot to use Local Cache to enable a seamless build-deploy-test lifecycle for staging if you get the benefits of Local Cache for the production slot.
- Test your site against your Staging slot.
- When you are ready, issue a [swap operation](#) between your Staging and Production slots.
- Sticky settings include name and sticky to a slot. So when the Staging slot gets swapped into Production, it inherits the Local Cache app settings. The newly swapped Production slot will run against the local cache after a few minutes and will be warmed up as part of slot warmup after swap. So when the slot swap is complete, your Production slot is running against the local cache.

## Frequently asked questions (FAQ)

### How can I tell if Local Cache applies to my app?

If your app needs a high-performance, reliable content store, does not use the content store to write critical data at runtime, and is less than 2 GB in total size, then the answer is "yes"! To get the total size of your /site and /siteextensions folders, you can use the site extension "Azure Web Apps Disk Usage."

### How can I tell if my site has switched to using Local Cache?

If you're using the Local Cache feature with Staging Environments, the swap operation does not complete until Local Cache is warmed up. To check if your site is running against Local Cache, you can check the worker process environment variable `WEBSITE_LOCALCACHE_READY`. Use the instructions on the [worker process environment variable](#) page to access the worker process environment variable on multiple instances.

## I just published new changes, but my app does not seem to have them. Why?

If your app uses Local Cache, then you need to restart your site to get the latest changes. Don't want to publish changes to a production site? See the slot options in the previous best practices section.

### NOTE

The [run from package](#) deployment option is not compatible with local cache.

## Where are my logs?

With Local Cache, your logs and data folders do look a little different. However, the structure of your subfolders remains the same, except that the subfolders are nestled under a subfolder with the format "unique VM identifier" + time stamp.

## I have Local Cache enabled, but my app still gets restarted. Why is that? I thought Local Cache helped with frequent app restarts.

Local Cache does help prevent storage-related app restarts. However, your app could still undergo restarts during planned infrastructure upgrades of the VM. The overall app restarts that you experience with Local Cache enabled should be fewer.

## Does Local Cache exclude any directories from being copied to the faster local drive?

As part of the step that copies the storage content, any folder that is named repository is excluded. This helps with scenarios where your site content may contain a source control repository that may not be needed in day to day operation of the app.

## How to flush the local cache logs after a site management operation?

To flush the local cache logs, stop and restart the app. This action clears the old cache.

## More resources

[Environment variables and app settings reference](#)

# Azure App Service diagnostics overview

11/2/2021 • 5 minutes to read • [Edit Online](#)

When you're running a web application, you want to be prepared for any issues that may arise, from 500 errors to your users telling you that your site is down. App Service diagnostics is an intelligent and interactive experience to help you troubleshoot your app with no configuration required. When you do run into issues with your app, App Service diagnostics points out what's wrong to guide you to the right information to more easily and quickly troubleshoot and resolve the issue.

Although this experience is most helpful when you're having issues with your app within the last 24 hours, all the diagnostic graphs are always available for you to analyze.

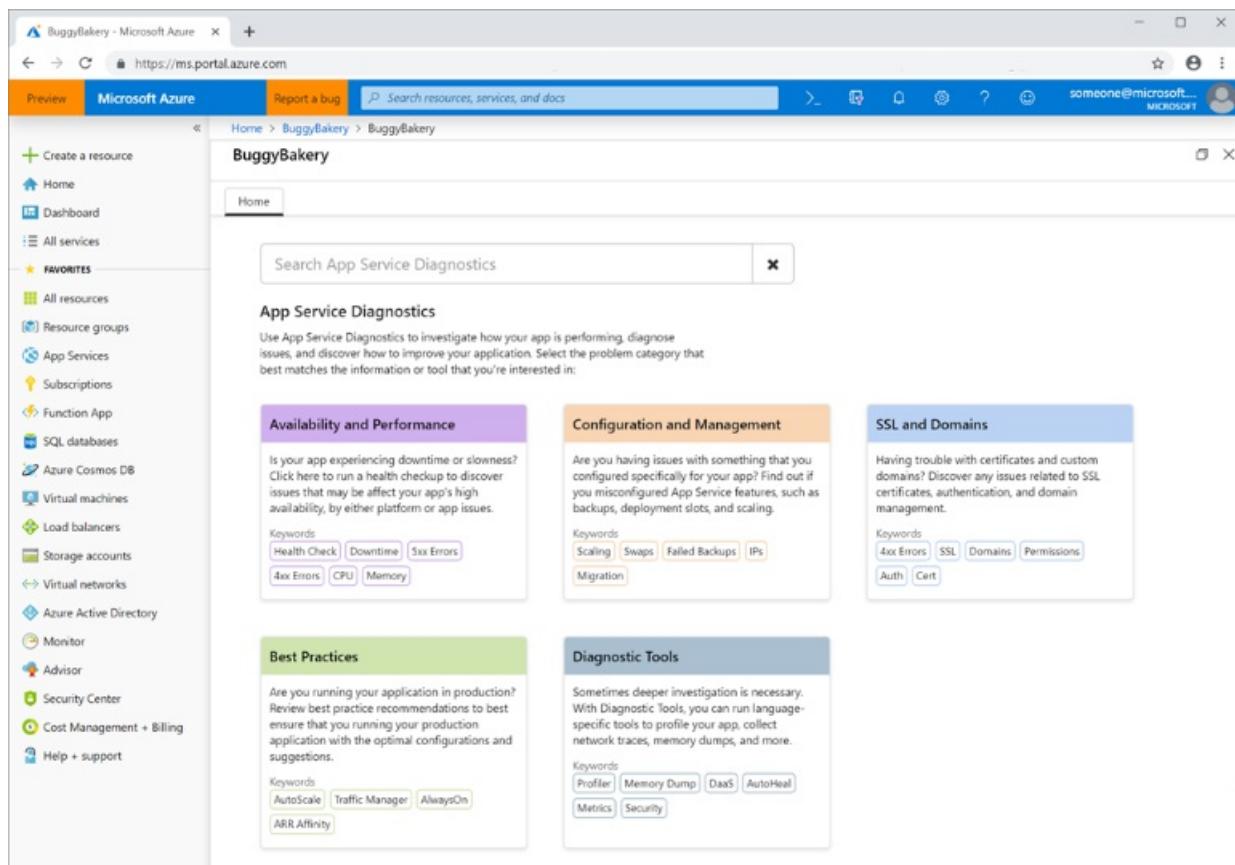
App Service diagnostics works for not only your app on Windows, but also apps on [Linux/containers](#), [App Service Environment](#), and [Azure Functions](#).

## Open App Service diagnostics

To access App Service diagnostics, navigate to your App Service web app or App Service Environment in the [Azure portal](#). In the left navigation, click on **Diagnose and solve problems**.

For Azure Functions, navigate to your function app, and in the top navigation, click on **Platform features**, and select **Diagnose and solve problems** from the **Resource management** section.

In the App Service diagnostics homepage, you can choose the category that best describes the issue with your app by using the keywords in each homepage tile. Also, this page is where you can find **Diagnostic Tools**. See [Diagnostic tools](#).



#### NOTE

If your app is down or performing slow, you can [collect a profiling trace](#) to identify the root cause of the issue. Profiling is light weight and is designed for production scenarios.

## Interactive interface

Once you select a homepage category that best aligns with your app's problem, App Service diagnostics' interactive interface, Genie, can guide you through diagnosing and solving problem with your app. You can use the tile shortcuts provided by Genie to view the full diagnostic report of the problem category that you are interested. The tile shortcuts provide you a direct way of accessing your diagnostic metrics.

The screenshot shows the 'Availability and Performance' tab selected in the navigation bar. A welcome message from Genie is displayed: 'Hello! Welcome to App Service Diagnostics! My name is Genie and I'm here to help you diagnose and solve problems.' Below this, a message says 'Here are some issues related to Availability and Performance that I can help with. Please select the tile that best describes your issue.' There are nine blue rectangular tiles arranged in three rows of three:

- Application Logs
- Container Issues
- CPU Usage
- Memory Usage
- Port Usage
- Process Full List
- Process List
- Web App Down
- Web App Restarted

After clicking on these tiles, you can see a list of topics related to the issue described in the tile. These topics provide snippets of notable information from the full report. You can click on any of these topics to investigate the issues further. Also, you can click on **View Full Report** to explore all the topics on a single page.

The screenshot shows the same interface after selecting the 'Wrong port is exposed: 5000 != 8080' tile. A message at the top says 'Okay give me a moment while I analyze your app for any issues related to this tile. Once the detectors load, feel free to click to investigate each topic further.' To the right, a blue button says 'I am interested in Application Logs'. Below, there is a list of topics with icons and arrows:

- Wrong port is exposed: 5000 != 8080 (highlighted with a red border)
- Verbose application logging is off.
- Link to Full Logs

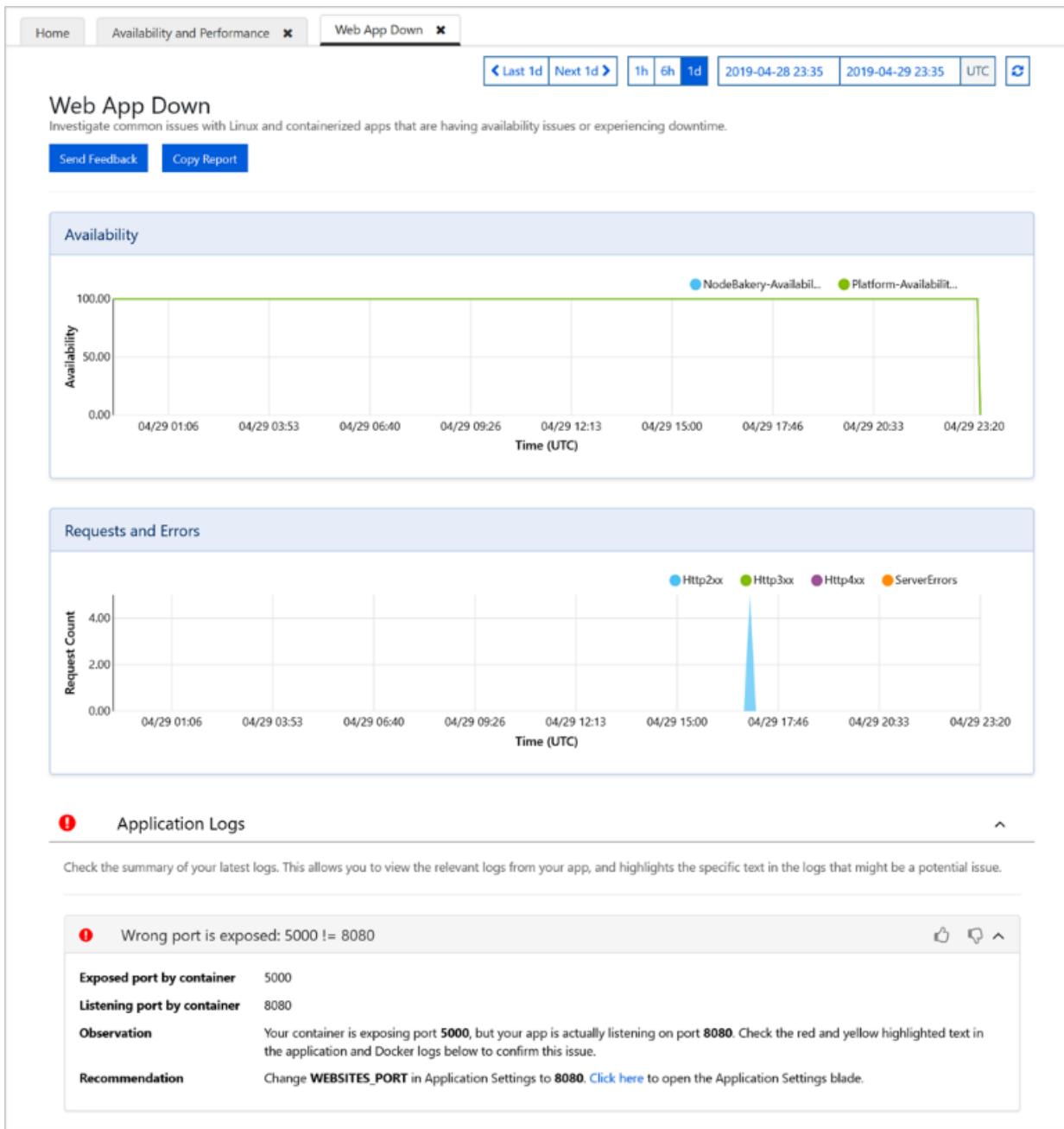
Web App Down

[View Full Report >](#)

- ! Application Logs >
- ! Container Issues >
- i CPU Usage >
- ✓ Memory Usage >
- i Port Usage >
- i Process List >
- ⚠ Web App Restarted >

## Diagnostic report

After you choose to investigate the issue further by clicking on a topic, you can view more details about the topic often supplemented with graphs and markdowns. Diagnostic report can be a powerful tool for pinpointing the problem with your app.



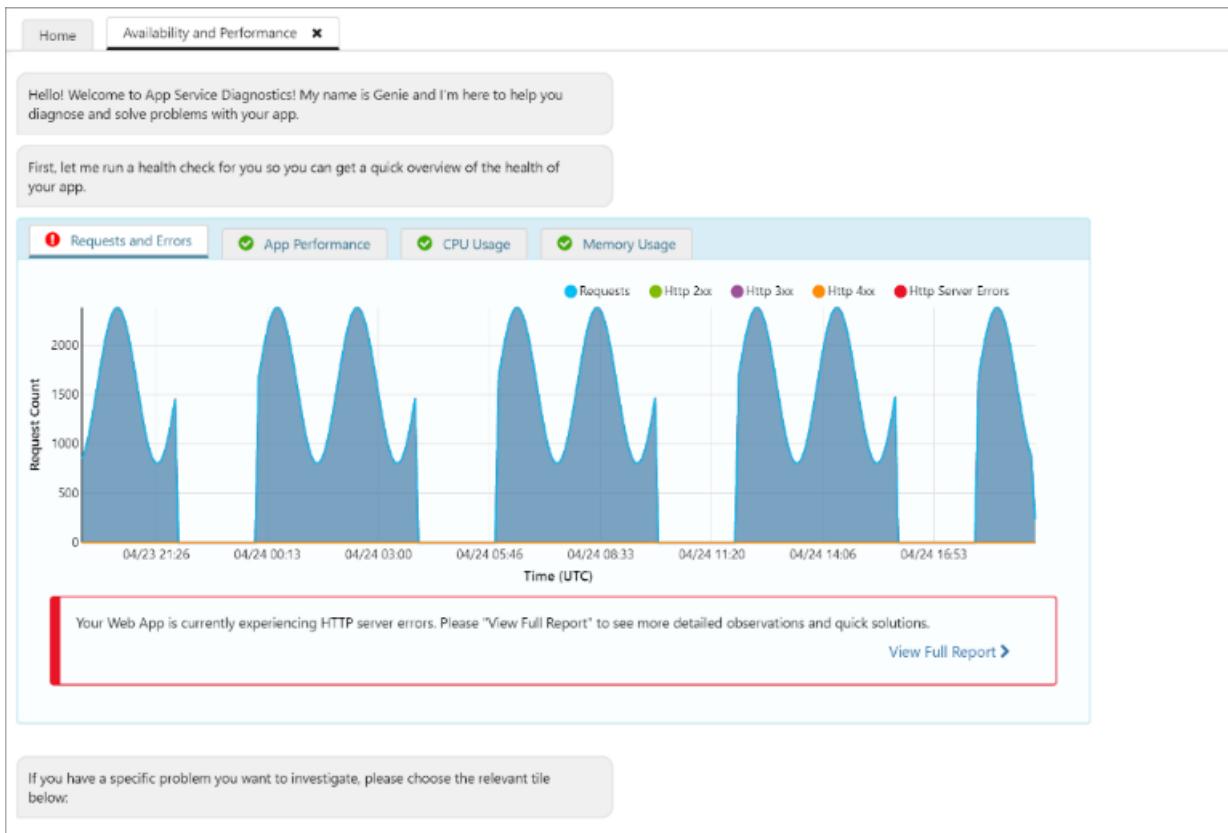
## Health checkup

If you don't know what's wrong with your app or don't know where to start troubleshooting your issues, the health checkup is a good place to start. The health checkup analyzes your applications to give you a quick, interactive overview that points out what's healthy and what's wrong, telling you where to look to investigate the issue. Its intelligent and interactive interface provides you with guidance through the troubleshooting process. Health checkup is integrated with the Genie experience for Windows apps and web app down diagnostic report for Linux apps.

### Health checkup graphs

There are four different graphs in the health checkup.

- **requests and errors:** A graph that shows the number of requests made over the last 24 hours along with HTTP server errors.
- **app performance:** A graph that shows response time over the last 24 hours for various percentile groups.
- **CPU usage:** A graph that shows the overall percent CPU usage per instance over the last 24 hours.
- **memory usage:** A graph that shows the overall percent physical memory usage per instance over the last 24 hours.



## Investigate application code issues (only for Windows app)

Because many app issues are related to issues in your application code, App Service diagnostics integrates with [Application Insights](#) to highlight exceptions and dependency issues to correlate with the selected downtime. Application Insights has to be enabled separately.

Application Insights		
Application Exceptions that occurred during this time period		
Message	Exception	Count
Value cannot be null. Parameter name: UPDATE_BREAD_INFO_EXTERNALLY	System.ArgumentNullException at BuildBakery.Controllers.HomeController.CheckUpdateNeeded	4374
<a href="#">View More in App Insights</a>		

To view Application Insights exceptions and dependencies, select the **web app down** or **web app slow** tile shortcuts.

## Troubleshooting steps (only for Windows app)

If an issue is detected with a specific problem category within the last 24 hours, you can view the full diagnostic report, and App Service diagnostics may prompt you to view more troubleshooting advice and next steps for a more guided experience.

**Troubleshooting and Next Steps**  
Next steps curated specifically for your app

**Scale Out App Service Plan**

**Mitigation**

**Remote Profile App**

**Investigation**

### Scale out your App Service Plan

**Mitigation**

Increase the number of the instances in your app service plan. The requests to your app will be spread across all instances.

**Current App Service Plan**

**S1 Standard**   **1**   **2**  
Tier      Instance Count      App Count

Your current App Service Plan has only 1 instance. We suggest scaling out to at least two instances to make sure that when we do infrastructure upgrades, your app has the best chance of being highly available.

**Open Scale Out App Service Plan**

**Why you should scale up**

- Your app is consistently using high levels of CPU on every instance.

## Diagnostic tools

Diagnostics Tools include more advanced diagnostic tools that help you investigate application code issues, slowness, connection strings, and more. and proactive tools that help you mitigate issues with CPU usage, requests, and memory.

### Proactive CPU monitoring (only for Windows app)

Proactive CPU monitoring provides you an easy, proactive way to take an action when your app or child process for your app is consuming high CPU resources. You can set your own CPU threshold rules to temporarily mitigate a high CPU issue until the real cause for the unexpected issue is found. For more information, see [Mitigate your CPU problems before they happen](#).

#### Proactive CPU Monitoring

Proactive CPU Monitoring provides you with an easy way to take an action when your app or any child process for your app is consuming high CPU resources. The triggers allow you to define CPU thresholds at which you want the actions to be taken. This feature also helps in mitigating the issue by killing the process consuming high CPU. Please note that these mitigations should only be considered a temporary workaround until you find the real cause for the issue causing the unexpected behavior.

##### - 1. Configure

Monitoring Enabled

##### CPU Threshold

This is the CPU threshold at which the rule will be triggered



##### Threshold Seconds

For the rule to trigger, CPU should exceed 75% for this many seconds



## Auto-healing

Auto-healing is a mitigation action you can take when your app is having unexpected behavior. You can set your own rules based on request count, slow request, memory limit, and HTTP status code to trigger mitigation actions. Use the tool to temporarily mitigate an unexpected behavior until you find the root cause. The tool is currently available for Windows Web Apps, Linux Web Apps, and Linux Custom Containers. Supported conditions and mitigation vary depending on the type of the web app. For more information, see [Announcing the new auto healing experience in app service diagnostics](#) and [Announcing Auto Heal for Linux](#).

Configure Mitigation Rules    ProActive Auto-Healing    View Auto-Healing History

Auto-Healing Enabled    **On**

1. Define Conditions

Request Duration    Memory Limit    Request Count    Status Codes

2. Configure Actions

Recycle Process    Log an Event    Custom Action

3. Override when Action executes (Optional)

Startup Time

4. Review and Save your Settings

Current Settings  
No rule configured!

**Save**    **Cancel**

### Proactive auto-healing (only for Windows app)

Like proactive CPU monitoring, proactive auto-healing is a turn-key solution to mitigating unexpected behavior of your app. Proactive auto-healing restarts your app when App Service determines that your app is in an unrecoverable state. For more information, see [Introducing Proactive Auto Heal](#).

### Navigator and change analysis (only for Windows app)

In a large team with continuous integration and where your app has many dependencies, it can be difficult to pinpoint the specific change that causes an unhealthy behavior. Navigator helps get visibility on your app's topology by automatically rendering a dependency map of your app and all the resources in the same subscription. Navigator lets you view a consolidated list of changes made by your app and its dependencies and narrow down on a change causing unhealthy behavior. It can be accessed through the homepage tile **Navigator** and needs to be enabled before you use it the first time. For more information, see [Get visibility into your app's dependencies with Navigator](#).



2 change groups have been detected for servers/navigator-sql

Changes were last scanned on Thu, Aug 8 2019, 9:14:12 am

[Go to Change Analysis Settings](#)

Click the below button to scan your resource and get the latest changes

[Scan changes now](#)

Properties				
Code				
	Thu 1 July 2019		Fri 2 August 2019	Sat 3

Level	Time	Name	Description	Initiated By
> 🚨	Jun 5 2019, 2:45:49 pm	\Areas\HelpPage\Views\Web.config	Application file	someone@microsoft.com
> 🚨	Jun 5 2019, 2:45:49 pm	\Views\Web.config	Application file	someone@microsoft.com
▼ 🛡️	Jun 5 2019, 2:45:49 pm	\Web.config	Application file	someone@microsoft.com
<pre> 1 &lt;?xml version="1.0" encoding="utf-8"?&gt; 2 &lt;!-- 3 For more information on how to configure your ASP.NET 4 https://go.microsoft.com/fwlink/?LinkId=301879 5 --&gt; 6 &lt;configuration&gt; 7   &lt;connectionStrings&gt; 8 -   &lt;add name="MyDbConnection" connectionString="Serv&lt;+ &lt;?xml version="1.0" encoding="utf-8"?&gt; 2 &lt;!-- 3 For more information on how to configure your ASP.NET 4 https://go.microsoft.com/fwlink/?LinkId=301879 5 --&gt; 6 &lt;configuration&gt; 7   &lt;connectionStrings&gt; 8 +   &lt;add name="MyDbConnection" connectionString="Serv 9   &lt;add name="StorageConnection" connectionString="E&lt;+ &lt;!-- 10  &lt;/connectionStrings&gt; 11  &lt;appSettings&gt; 12    &lt;add key="webpages:Version" value="3.0.0.0" /&gt; 13    &lt;add key="webpages:Enabled" value="false" /&gt; </pre>				
> 🚨	Jun 5 2019, 2:45:49 pm	\bin\changeanalysis-webapp.dll	Application file	someone@microsoft.com
> 🚨	Jun 5 2019, 2:45:49 pm	\bin\changeanalysis-webapp.pdb	Application file	someone@microsoft.com

Change analysis for app changes can be accessed through tile shortcuts, [Application Changes](#) and [Application Crashes in Availability and Performance](#) so you can use it concurrently with other metrics. Before using the feature, you must first enable it. For more information, see [Announcing the new change analysis experience in App Service Diagnostics](#).

Post your questions or feedback at [UserVoice](#) by adding "[Diag]" in the title.

# Configure an App Service app in the Azure portal

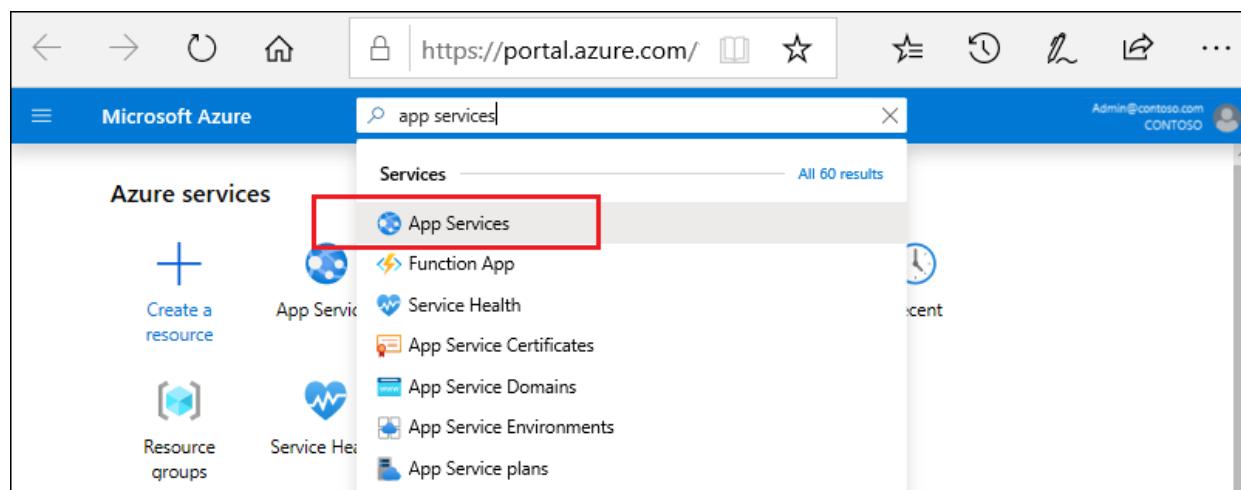
11/2/2021 • 9 minutes to read • [Edit Online](#)

This article explains how to configure common settings for web apps, mobile back end, or API app using the [Azure portal](#).

## Configure app settings

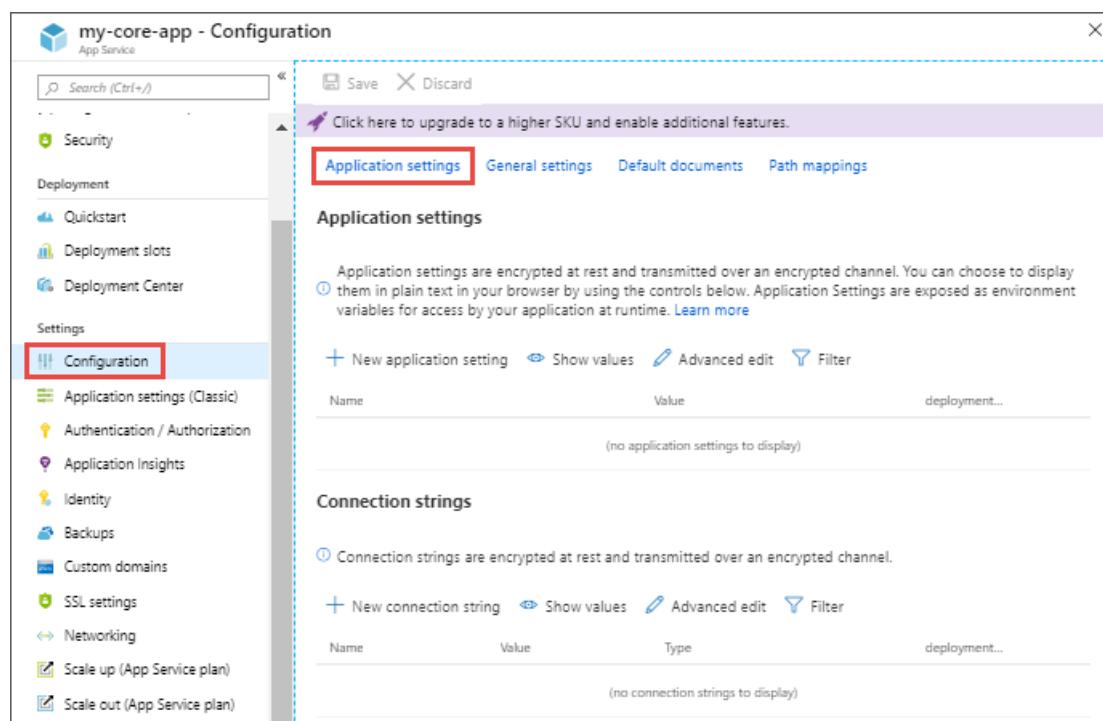
In App Service, app settings are variables passed as environment variables to the application code. For Linux apps and custom containers, App Service passes app settings to the container using the `--env` flag to set the environment variable in the container. In either case, they're injected into your app environment at app startup. When you add, remove, or edit app settings, App Service triggers an app restart. App setting names can't contain periods (.). If an app setting contains a period, the period is replaced with an underscore in the container.

In the [Azure portal](#), search for and select **App Services**, and then select your app.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with icons for back, forward, refresh, and home. The URL is https://portal.azure.com/. Below the navigation bar is a search bar containing 'app services'. On the left, there's a sidebar titled 'Azure services' with options for 'Create a resource', 'Resource groups', and 'Service Health'. The main area is titled 'Services' and shows a list of items under 'All 60 results': App Services (highlighted with a red box), Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, and App Service plans. The 'App Services' item has a blue icon with a white gear and a blue circle.

In the app's left menu, select **Configuration > Application settings**.



The screenshot shows the configuration page for an app named 'my-core-app'. The left sidebar has a 'Configuration' section selected and highlighted with a red box. Other options in the sidebar include Security, Deployment (Quickstart, Deployment slots, Deployment Center), and Settings (Application settings (Classic), Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, SSL settings, Networking, Scale up (App Service plan), Scale out (App Service plan)). The main content area has tabs for Application settings, General settings, Default documents, and Path mappings. The 'Application settings' tab is selected and highlighted with a red box. It contains a note about encrypted transmission and a table for adding new application settings. The 'Connection strings' section also contains a note about encryption and a table for managing connection strings. Buttons for 'Save' and 'Discard' are at the top right.

For ASP.NET and ASP.NET Core developers, setting app settings in App Service are like setting them in `<appSettings>` in *Web.config* or *appsettings.json*, but the values in App Service override the ones in *Web.config* or *appsettings.json*. You can keep development settings (for example, local MySQL password) in *Web.config* or *appsettings.json* and production secrets (for example, Azure MySQL database password) safely in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

Other language stacks, likewise, get the app settings as environment variables at runtime. For language-stack specific steps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Ruby](#)
- [Custom containers](#)

App settings are always encrypted when stored (encrypted-at-rest).

#### NOTE

App settings can also be resolved from [Key Vault](#) using [Key Vault references](#).

### Show hidden values

By default, values for app settings are hidden in the portal for security. To see a hidden value of an app setting, click the **Value** field of that setting. To see the values of all app settings, click the **Show value** button.

### Add or edit

To add a new app setting, click **New application setting**. In the dialog, you can [stick the setting to the current slot](#).

To edit a setting, click the **Edit** button on the right side.

When finished, click **Update**. Don't forget to click **Save** back in the [Configuration](#) page.

#### NOTE

In a default Linux app service or a custom Linux container, any nested JSON key structure in the app setting name like `ApplicationInsights:InstrumentationKey` needs to be configured in App Service as `ApplicationInsights__InstrumentationKey` for the key name. In other words, any `:` should be replaced by `_` (double underscore).

### Edit in bulk

To add or edit app settings in bulk, click the **Advanced edit** button. When finished, click **Update**. Don't forget to click **Save** back in the [Configuration](#) page.

App settings have the following JSON formatting:

```
[
 {
 "name": "<key-1>",
 "value": "<value-1>",
 "slotSetting": false
 },
 {
 "name": "<key-2>",
 "value": "<value-2>",
 "slotSetting": false
 },
 ...
]
```

## Automate app settings with the Azure CLI

You can use the Azure CLI to create and manage settings from the command line.

- Assign a value to a setting with [az webapp config app settings set](#):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
<setting-name>=<value>
```

Replace `<setting-name>` with the name of the setting, and `<value>` with the value to assign to it. This command creates the setting if it doesn't already exist.

- Show all settings and their values with [az webapp config appsettings list](#):

```
az webapp config appsettings list --name <app-name> --resource-group <resource-group-name>
```

- Remove one or more settings with [az webapp config app settings delete](#):

```
az webapp config appsettings delete --name <app-name> --resource-group <resource-group-name> --
setting-names {<names>}
```

Replace `<names>` with a space-separated list of setting names.

## Configure connection strings

In the [Azure portal](#), search for and select **App Services**, and then select your app. In the app's left menu, select **Configuration > Application settings**.

For ASP.NET and ASP.NET Core developers, setting connection strings in App Service are like setting them in `<connectionStrings>` in *Web.config*, but the values you set in App Service override the ones in *Web.config*. You can keep development settings (for example, a database file) in *Web.config* and production secrets (for example, SQL Database credentials) safely in App Service. The same code uses your development settings when you debug locally, and it uses your production secrets when deployed to Azure.

For other language stacks, it's better to use [app settings](#) instead, because connection strings require special formatting in the variable keys in order to access the values.

#### NOTE

There is one case where you may want to use connection strings instead of app settings for non-.NET languages: certain Azure database types are backed up along with the app *only* if you configure a connection string for the database in your App Service app. For more information, see [What gets backed up](#). If you don't need this automated backup, then use app settings.

At runtime, connection strings are available as environment variables, prefixed with the following connection types:

- SQLServer: `SQLCONNSTR_`
- MySQL: `MYSQLCONNSTR_`
- SQLAzure: `SQLAZURECONNSTR_`
- Custom: `CUSTOMCONNSTR_`
- PostgreSQL: `POSTGRESQLCONNSTR_`

For example, a MySQL connection string named *connectionstring1* can be accessed as the environment variable `MYSQLCONNSTR_connectionString1`. For language-stack specific steps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Ruby](#)

- [Custom containers](#)

Connection strings are always encrypted when stored (encrypted-at-rest).

**NOTE**

Connection strings can also be resolved from [Key Vault](#) using [Key Vault references](#).

### Show hidden values

By default, values for connection strings are hidden in the portal for security. To see a hidden value of a connection string, just click the **Value** field of that string. To see the values of all connection strings, click the **Show value** button.

### Add or edit

To add a new connection string, click **New connection string**. In the dialog, you can [stick the connection string to the current slot](#).

To edit a setting, click the **Edit** button on the right side.

When finished, click **Update**. Don't forget to click **Save** back in the **Configuration** page.

### Edit in bulk

To add or edit connection strings in bulk, click the **Advanced edit** button. When finished, click **Update**. Don't forget to click **Save** back in the **Configuration** page.

Connection strings have the following JSON formatting:

```
[
 {
 "name": "name-1",
 "value": "conn-string-1",
 "type": "SQLServer",
 "slotSetting": false
 },
 {
 "name": "name-2",
 "value": "conn-string-2",
 "type": "PostgreSQL",
 "slotSetting": false
 },
 ...
]
```

## Configure general settings

In the [Azure portal](#), search for and select **App Services**, and then select your app. In the app's left menu, select **Configuration > General settings**.

my-core-app | Configuration

App Service

Search (Ctrl+ /) Refresh Save Discard

Application settings General settings (selected) Default documents Path mappings

Security Events (preview)

Deployment

- Quickstart
- Deployment slots
- Deployment Center
- Deployment Center (Preview)

Settings

- Configuration (selected)
- Authentication / Authorization
- Application Insights
- Identity
- Backups
- Custom domains

Stack settings

Stack: .NET Core

Platform settings

Platform: 32 Bit

Managed pipeline version: Integrated

FTP state: All allowed

Here, you can configure some common settings for the app. Some settings require you to [scale up to higher pricing tiers](#).

- **Stack settings:** The software stack to run the app, including the language and SDK versions.

For Linux apps and custom container apps, you can select the language runtime version and set an optional **Startup command** or a startup command file.

Stack settings

Stack: Python

Major version: Python 3

Minor version: Python 3.8

Startup Command:

Provide an optional startup command that will be run as part of container startup. [Learn more](#)

- **Platform settings:** Lets you configure settings for the hosting platform, including:
  - **Bitness:** 32-bit or 64-bit. (Defaults to 32-bit for App Service created in the portal.)
  - **WebSocket protocol:** For [ASP.NET SignalR](#) or [socket.io](#), for example.
  - **Always On:** Keeps the app loaded even when there's no traffic. When **Always On** is not turned on (default), the app is unloaded after 20 minutes without any incoming requests. The unloaded app can cause high latency for new requests because of its warm-up time. When **Always On** is turned on, the front-end load balancer sends a GET request to the application root every five minutes. The continuous ping prevents the app from being unloaded.

Always On is required for continuous WebJobs or for WebJobs that are triggered using a CRON expression.

- **Managed pipeline version:** The IIS [pipeline mode](#). Set it to **Classic** if you have a legacy app that requires an older version of IIS.
- **HTTP version:** Set to **2.0** to enable support for [HTTPS/2](#) protocol.

#### NOTE

Most modern browsers support HTTP/2 protocol over TLS only, while non-encrypted traffic continues to use HTTP/1.1. To ensure that client browsers connect to your app with HTTP/2, secure your custom DNS name. For more information, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

- **ARR affinity:** In a multi-instance deployment, ensure that the client is routed to the same instance for the life of the session. You can set this option to **Off** for stateless applications.
- **Debugging:** Enable remote debugging for [ASP.NET](#), [ASP.NET Core](#), or [Node.js](#) apps. This option turns off automatically after 48 hours.
- **Incoming client certificates:** require client certificates in [mutual authentication](#).

## Configure default documents

This setting is only for Windows apps.

In the [Azure portal](#), search for and select **App Services**, and then select your app. In the app's left menu, select **Configuration > Default documents**.

The screenshot shows the Azure portal configuration interface for an app named "my-core-app". The left sidebar has a tree view with "Configuration" selected. The main content area has tabs for "Application settings", "General settings", "Default documents" (which is highlighted with a red box), and "Path mappings". Under "Application settings", there is a note about encrypted application settings and a table for managing them. Under "Connection strings", there is a note about encrypted connection strings and a table for managing them.

The default document is the web page that's displayed at the root URL for a website. The first matching file in the list is used. To add a new default document, click **New document**. Don't forget to click **Save**.

If the app uses modules that route based on URL instead of serving static content, there is no need for default documents.

## Configure path mappings

In the [Azure portal](#), search for and select **App Services**, and then select your app. In the app's left menu, select **Configuration > Path mappings**.

The screenshot shows the Azure portal's Configuration blade for an App Service named "my-core-app". The "Path mappings" tab is selected. The left sidebar lists various configuration options like Security, Deployment, and Settings, with "Configuration" being the active tab. The main content area contains sections for Application settings and Connection strings, both of which are currently empty.

#### NOTE

The Path mappings tab may display OS-specific settings that differ from the example shown here.

### Windows apps (uncontainerized)

For Windows apps, you can customize the IIS handler mappings and virtual applications and directories.

Handler mappings let you add custom script processors to handle requests for specific file extensions. To add a custom handler, click **New handler mapping**. Configure the handler as follows:

- **Extension.** The file extension you want to handle, such as `*.php` or `handler.cgi`.
- **Script processor.** The absolute path of the script processor to you. Requests to files that match the file extension are processed by the script processor. Use the path `D:\home\site\wwwroot` to refer to your app's root directory.
- **Arguments.** Optional command-line arguments for the script processor.

Each app has the default root path (`/`) mapped to `D:\home\site\wwwroot`, where your code is deployed by default. If your app root is in a different folder, or if your repository has more than one application, you can edit or add virtual applications and directories here.

From the Path mappings tab, click **New virtual application or directory**.

- To map a virtual directory to a physical path, leave the **Directory** check box selected. Specify the virtual directory and the corresponding relative (physical) path to the website root (`D:\home`).
- To mark a virtual directory as a web application, clear the **Directory** check box.

The screenshot shows the 'livewire | Configuration' blade in the Azure portal. On the left, there's a navigation menu with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Events (preview), Quickstart, Deployment slots, Deployment Center, Deployment Center (Preview), Configuration, and Authentication / Authorization. The 'Configuration' link is highlighted. The main area has tabs for Application settings, General settings, and Default doc. Under Application settings, there's a 'Handler mappings' section with a 'New handler mapping' button. Below it is a table for 'Virtual applications and directories' with one entry: Virtual path '/' and Physical Path 'site\wwwroot\'. To the right, a modal window titled 'New application or directory' shows fields for 'Virtual path' and 'Physical Path', and a checkbox labeled 'Directory' which is checked and highlighted with a red box.

## Containerized apps

You can [add custom storage for your containerized app](#). Containerized apps include all Linux apps and also the Windows and Linux custom containers running on App Service. Click **New Azure Storage Mount** and configure your custom storage as follows:

- **Name:** The display name.
- **Configuration options:** Basic or Advanced.
- **Storage accounts:** The storage account with the container you want.
- **Storage type:** Azure Blobs or Azure Files.

### NOTE

Windows container apps only support Azure Files.

- **Storage container:** For basic configuration, the container you want.
- **Share name:** For advanced configuration, the file share name.
- **Access key:** For advanced configuration, the access key.
- **Mount path:** The absolute path in your container to mount the custom storage.

For more information, see [Access Azure Storage as a network share from a container in App Service](#).

## Configure language stack settings

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Ruby](#)

## Configure custom containers

See [Configure a custom Linux container for Azure App Service](#)

## Next steps

- [Environment variables and app settings reference](#)
- [Configure a custom domain name in Azure App Service](#)
- [Set up staging environments in Azure App Service](#)
- [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)
- [Enable diagnostic logs](#)
- [Scale an app in Azure App Service](#)
- [Monitoring basics in Azure App Service](#)
- [Change applicationHost.config settings with applicationHost.xdt](#)

# Configure an ASP.NET app for Azure App Service

11/2/2021 • 3 minutes to read • [Edit Online](#)

## NOTE

For ASP.NET Core, see [Configure an ASP.NET Core app for Azure App Service](#)

ASP.NET apps must be deployed to Azure App Service as compiled binaries. The Visual Studio publishing tool builds the solution and then deploys the compiled binaries directly, whereas the App Service deployment engine deploys the code repository first and then compiles the binaries.

This guide provides key concepts and instructions for ASP.NET developers. If you've never used Azure App Service, follow the [ASP.NET quickstart](#) and [ASP.NET with SQL Database tutorial](#) first.

## Show supported .NET Framework runtime versions

In App Service, the Windows instances already have all the supported .NET Framework versions installed. To show the .NET Framework runtime and SDK versions available to you, navigate to

`https://<app-name>.scm.azurewebsites.net/DebugConsole` and run the appropriate command in the browser-based console:

For CLR 4 runtime versions (.NET Framework 4 and above):

```
ls "D:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\.NETFramework\"
```

Latest .NET Framework version may not be immediately available.

For CLR 2 runtime versions (.NET Framework 3.5 and below):

```
ls "D:\Program Files (x86)\Reference Assemblies\Microsoft\Framework\"
```

## Show current .NET Framework runtime version

Run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query netFrameworkVersion
```

A value of `v4.0` means the latest CLR 4 version (.NET Framework 4.x) is used. A value of `v2.0` means a CLR 2 version (.NET Framework 3.5) is used.

## Set .NET Framework runtime version

By default, App Service uses the latest supported .NET Framework version to run your ASP.NET app. To run your app using .NET Framework 3.5 instead, run the following command in the [Cloud Shell](#) (v2.0 signifies CLR 2):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --net-framework-version v2.0
```

## Access environment variables

In App Service, you can [set app settings](#) and connection strings outside of your app code. Then you can access them in any class using the standard ASP.NET pattern:

```
using System.Configuration;
...
// Get an app setting
ConfigurationManager.AppSettings["MySetting"];
// Get a connection string
ConfigurationManager.ConnectionStrings["MyConnection"];
}
```

If you configure an app setting with the same name in App Service and in *web.config*, the App Service value takes precedence over the *web.config* value. The local *web.config* value lets you debug the app locally, but the App Service value lets you run the app in product with production settings. Connection strings work in the same way. This way, you can keep your application secrets outside of your code repository and access the appropriate values without changing your code.

## Deploy multi-project solutions

When a Visual Studio solution includes multiple projects, the Visual Studio publish process already includes selecting the project to deploy. When you deploy to the App Service deployment engine, such as with Git, or with ZIP deploy [with build automation enabled](#), the App Service deployment engine picks the first Web Site or Web Application Project it finds as the App Service app. You can specify which project App Service should use by specifying the `PROJECT` app setting. For example, run the following in the [Cloud Shell](#):

```
az webapp config appsettings set --resource-group <resource-group-name> --name <app-name> --settings
PROJECT=<project-name>/<project-name>.csproj"
```

## Get detailed exceptions page

When your ASP.NET app generates an exception in the Visual Studio debugger, the browser displays a detailed exception page, but in App Service that page is replaced by a generic error message. To display the detailed exception page in App Service, open the *Web.config* file and add the `<customErrors mode="Off"/>` element under the `<system.web>` element. For example:

```
<system.web>
 <customErrors mode="Off"/>
</system.web>
```

Redeploy your app with the updated *Web.config*. You should now see the same detailed exception page.

## Access diagnostic logs

You can add diagnostic messages in your application code using [System.Diagnostics.Trace](#). For example:

```
Trace.TraceError("Record not found!"); // Error trace
Trace.TraceWarning("Possible data loss"); // Warning trace
Trace.TraceInformation("GET /Home/Index"); // Information trace
```

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging
filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

**NOTE**

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

## More resources

- [Tutorial: Build an ASP.NET app in Azure with SQL Database](#)
- [Environment variables and app settings reference](#)

# Configure an ASP.NET Core app for Azure App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

## NOTE

For ASP.NET in .NET Framework, see [Configure an ASP.NET app for Azure App Service](#)

ASP.NET Core apps must be deployed to Azure App Service as compiled binaries. The Visual Studio publishing tool builds the solution and then deploys the compiled binaries directly, whereas the App Service deployment engine deploys the code repository first and then compiles the binaries.

This guide provides key concepts and instructions for ASP.NET Core developers. If you've never used Azure App Service, follow the [ASP.NET Core quickstart](#) and [ASP.NET Core with SQL Database tutorial](#) first.

## Show supported .NET Core runtime versions

In App Service, the Windows instances already have all the supported .NET Core versions installed. To show the .NET Core runtime and SDK versions available to you, navigate to

`https://<app-name>.scm.azurewebsites.net/DebugConsole` and run the following command in the browser-based console:

```
dotnet --info
```

## Show .NET Core version

To show the current .NET Core version, run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query linuxFxVersion
```

To show all supported .NET Core versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes --linux | grep DOTNETCORE
```

## Set .NET Core version

Set the target framework in the project file for your ASP.NET Core project. For more information, see [Select the .NET Core version to use](#) in .NET Core documentation.

Run the following command in the [Cloud Shell](#) to set the .NET Core version to 3.1:

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --linux-fx-version "DOTNETCORE|3.1"
```

## Customize build automation

If you deploy your app using Git, or zip packages [with build automation enabled](#), the App Service build automation steps through the following sequence:

1. Run custom script if specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `dotnet restore` to restore NuGet dependencies.
3. Run `dotnet publish` to build a binary for production.
4. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run pre-build commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example specifies the two variables to a series of commands, separated by commas.

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
PRE_BUILD_COMMAND="echo foo, scripts/prebuild.sh"
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
POST_BUILD_COMMAND="echo foo, scripts/postbuild.sh"
```

For additional environment variables to customize build automation, see [Oryx configuration](#).

For more information on how App Service runs and builds ASP.NET Core apps in Linux, see [Oryx documentation: How .NET Core apps are detected and built](#).

## Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them in any class using the standard ASP.NET Core dependency injection pattern:

```
using Microsoft.Extensions.Configuration;

namespace SomeNamespace
{
 public class SomeClass
 {
 private IConfiguration _configuration;

 public SomeClass(IConfiguration configuration)
 {
 _configuration = configuration;
 }

 public SomeMethod()
 {
 // retrieve nested App Service app setting
 var myHierarchicalConfig = _configuration["My:Hierarchical:Config:Data"];
 // retrieve App Service connection string
 var myConnString = _configuration.GetConnectionString("MyDbConnection");
 }
 }
}
```

If you configure an app setting with the same name in App Service and in `appsettings.json`, for example, the App Service value takes precedence over the `appsettings.json` value. The local `appsettings.json` value lets you debug the app locally, but the App Service value lets you run the app in production with production settings.

Connection strings work in the same way. This way, you can keep your application secrets outside of your code repository and access the appropriate values without changing your code.

## NOTE

Note the [hierarchical configuration data](#) in `appsettings.json` is accessed using the `:` delimiter that's standard to .NET Core. To override a specific hierarchical configuration setting in App Service, set the app setting name with the same delimited format in the key. you can run the following example in the [Cloud Shell](#):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings My:Hierarchical:Config:Data="some value"
```

## Deploy multi-project solutions

When a Visual Studio solution includes multiple projects, the Visual Studio publish process already includes selecting the project to deploy. When you deploy to the App Service deployment engine, such as with Git, or with ZIP deploy [with build automation enabled](#), the App Service deployment engine picks the first Web Site or Web Application Project it finds as the App Service app. You can specify which project App Service should use by specifying the `PROJECT` app setting. For example, run the following in the [Cloud Shell](#):

```
az webapp config appsettings set --resource-group <resource-group-name> --name <app-name> --settings PROJECT="<>project-name/>project-name.csproj"
```

## Access diagnostic logs

ASP.NET Core provides a [built-in logging provider for App Service](#). In `Program.cs` of your project, add the provider to your application through the `ConfigureLogging` extension method, as shown in the following example:

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
 Host.CreateDefaultBuilder(args)
 .ConfigureLogging(logging =>
 {
 logging.AddAzureWebAppDiagnostics();
 })
 .ConfigureWebHostDefaults(webBuilder =>
 {
 webBuilder.UseStartup<Startup>();
 });
});
```

You can then configure and generate logs with the [standard .NET Core pattern](#).

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

**NOTE**

You can also inspect the log files from the browser at `https://<app-name>.scm.azurewebsites.net/api/logs/docker`.

To stop log streaming at any time, type `Ctrl + C`.

For more information on troubleshooting ASP.NET Core apps in App Service, see [Troubleshoot ASP.NET Core on Azure App Service and IIS](#)

## Get detailed exceptions page

When your ASP.NET Core app generates an exception in the Visual Studio debugger, the browser displays a detailed exception page, but in App Service that page is replaced by a generic **HTTP 500** error or **An error occurred while processing your request**. message. To display the detailed exception page in App Service, Add the `ASPNETCORE_ENVIRONMENT` app setting to your app by running the following command in the [Cloud Shell](#).

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
ASPNETCORE_ENVIRONMENT="Development"
```

## Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to know if the user requests are encrypted or not, configure the Forwarded Headers Middleware in *Startup.cs*.

- Configure the middleware with `ForwardedHeadersOptions` to forward the `X-Forwarded-For` and `X-Forwarded-Proto` headers in `Startup.ConfigureServices`.
- Add private IP address ranges to the known networks, so that the middleware can trust the App Service load balancer.
- Invoke the `UseForwardedHeaders` method in `Startup.Configure` before calling other middleware.

Putting all three elements together, your code looks like the following example:

```
public void ConfigureServices(IServiceCollection services)
{
 services.AddMvc();

 services.Configure<ForwardedHeadersOptions>(options =>
 {
 options.ForwardedHeaders =
 ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto;
 // These three subnets encapsulate the applicable Azure subnets. At the moment, it's not possible to
 // narrow it down further.
 options.KnownNetworks.Add(new IPNetwork(IPAddress.Parse("::ffff:10.0.0.0"), 104));
 options.KnownNetworks.Add(new IPNetwork(IPAddress.Parse("::ffff:192.168.0.0"), 112));
 options.KnownNetworks.Add(new IPNetwork(IPAddress.Parse("::ffff:172.16.0.0"), 108));
 });
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
 app.UseForwardedHeaders();

 ...

 app.UseMvc();
}
```

For more information, see [Configure ASP.NET Core to work with proxy servers and load balancers](#).

## Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace <app-name> with your app name:

```
https://<app-name>.scm.azurewebsites.net/webssh/host
```

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

```
root@9e933156516f:~# service --status-all
[+] apache2
[-] bootlogs
[-] bootmisc.sh
[-] checkfs.sh
[-] checkroot-bootclean.sh
[-] checkroot.sh
[-] hostname.sh
[?] hwclock.sh
[-] killprocs
[-] motd
[-] mountall-bootclean.sh
[-] mountall.sh
[-] mountdevsubfs.sh
[-] mountkernfs.sh
[-] mountnfs-bootclean.sh
[-] mountnfs.sh
[-] mysql
[-] procps
[-] rc.local
[-] rmmlogin
[-] sendsigs
[+] ssh
[+] udev
[?] udev-finish
[-] umountfs
[-] umountnfs.sh
[-] umountroot
[-] urandom
root@9e933156516f:~#
```

ssh://root@ 2222 SSH CONNECTION ESTABLISHED

#### NOTE

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

## robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415
"_" "_"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

## Next steps

[Tutorial: ASP.NET Core app with SQL Database](#)

[App Service Linux FAQ](#)

Or, see additional resources:

[Environment variables and app settings reference](#)

# Configure a Node.js app for Azure App Service

11/2/2021 • 11 minutes to read • [Edit Online](#)

Node.js apps must be deployed with all the required NPM dependencies. The App Service deployment engine automatically runs `npm install --production` for you when you deploy a [Git repository](#), or a [Zip package with build automation enabled](#). If you deploy your files using [FTP/S](#), however, you need to upload the required packages manually.

This guide provides key concepts and instructions for Node.js developers who deploy to App Service. If you've never used Azure App Service, follow the [Node.js quickstart](#) and [Node.js with MongoDB tutorial](#) first.

## Show Node.js version

To show the current Node.js version, run the following command in the [Cloud Shell](#):

```
az webapp config appsettings list --name <app-name> --resource-group <resource-group-name> --query "[? name=='WEBSITE_NODE_DEFAULT_VERSION'].value"
```

To show all supported Node.js versions, navigate to

<https://<sitename>.scm.azurewebsites.net/api/diagnostics/runtime> or run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes | grep node
```

To show the current Node.js version, run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query linuxFxVersion
```

To show all supported Node.js versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes --linux | grep NODE
```

## Set Node.js version

To set your app to a [supported Node.js version](#), run the following command in the [Cloud Shell](#) to set `WEBSITE_NODE_DEFAULT_VERSION` to a supported version:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings WEBSITE_NODE_DEFAULT_VERSION="10.15"
```

This setting specifies the Node.js version to use, both at runtime and during automated package restore during App Service build automation. This setting only recognizes major minor versions, the *LTS* moniker is not supported.

#### NOTE

You should set the Node.js version in your project's `package.json`. The deployment engine runs in a separate process that contains all the supported Node.js versions.

To set your app to a [supported Node.js version](#), run the following command in the [Cloud Shell](#):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "NODE|10.14"
```

This setting specifies the Node.js version to use, both at runtime and during automated package restore in Kudu.

#### NOTE

You should set the Node.js version in your project's `package.json`. The deployment engine runs in a separate container that contains all the supported Node.js versions.

## Get port number

Your Node.js app needs to listen to the right port to receive incoming requests.

In App Service on Windows, Node.js apps are hosted with [IISNode](#), and your Node.js app should listen to the port specified in the `process.env.PORT` variable. The following example shows how you do it in a simple Express app:

App Service sets the environment variable `PORT` in the Node.js container, and forwards the incoming requests to your container at that port number. To receive the requests, your app should listen to that port using `process.env.PORT`. The following example shows how you do it in a simple Express app:

```
const express = require('express')
const app = express()
const port = process.env.PORT || 3000

app.get('/', (req, res) => {
 res.send('Hello World!')
})

app.listen(port, () => {
 console.log(`Example app listening at http://localhost:${port}`)
})
```

## Customize build automation

If you deploy your app using Git, or zip packages [with build automation enabled](#), the App Service build automation steps through the following sequence:

1. Run custom script if specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `npm install` without any flags, which includes npm `preinstall` and `postinstall` scripts and also installs `devDependencies`.
3. Run `npm run build` if a build script is specified in your `package.json`.
4. Run `npm run build:azure` if a build:azure script is specified in your `package.json`.
5. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

#### NOTE

As described in [npm docs](#), scripts named `prebuild` and `postbuild` run before and after `build`, respectively, if specified. `preinstall` and `postinstall` run before and after `install`, respectively.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run pre-build commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example specifies the two variables to a series of commands, separated by commas.

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
PRE_BUILD_COMMAND="echo foo, scripts/prebuild.sh"
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
POST_BUILD_COMMAND="echo foo, scripts/postbuild.sh"
```

For additional environment variables to customize build automation, see [Oryx configuration](#).

For more information on how App Service runs and builds Node.js apps in Linux, see [Oryx documentation: How Nodejs apps are detected and built](#).

## Configure Node.js server

The Node.js containers come with [PM2](#), a production process manager. You can configure your app to start with PM2, or with NPM, or with a custom command.

TOOL	PURPOSE
Run with PM2	<b>Recommended</b> - Production or staging use. PM2 provides a full-service app management platform.
Run npm start	Development use only.
Run custom command	Either development or staging.

### Run with PM2

The container automatically starts your app with PM2 when one of the common Node.js files is found in your project:

- `bin/www`
- `server.js`
- `app.js`
- `index.js`
- `hostingstart.js`
- One of the following [PM2 files](#): `process.json` and `ecosystem.config.js`

You can also configure a custom start file with the following extensions:

- A `.js` file
- A [PM2 file](#) with the extension `.json`, `.config.js`, `.yaml`, or `.yml`

To add a custom start file, run the following command in the [Cloud Shell](#):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<filename-with-extension>"
```

## Run custom command

App Service can start your app using a custom command, such as an executable like *run.sh*. For example, to run `npm run start:prod`, run the following command in the [Cloud Shell](#):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "npm run start:prod"
```

## Run npm start

To start your app using `npm start`, just make sure a `start` script is in the *package.json* file. For example:

```
{
 ...
 "scripts": {
 "start": "gulp",
 ...
 },
 ...
}
```

To use a custom *package.json* in your project, run the following command in the [Cloud Shell](#):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<filename>.json"
```

## Debug remotely

### NOTE

Remote debugging is currently in Preview.

You can debug your Node.js app remotely in [Visual Studio Code](#) if you configure it to [run with PM2](#), except when you run it using a `*.config.js`, `*.yml`, or `.yaml`.

In most cases, no extra configuration is required for your app. If your app is run with a *process.json* file (default or custom), it must have a `script` property in the JSON root. For example:

```
{
 "name" : "worker",
 "script" : "./index.js",
 ...
}
```

To set up Visual Studio Code for remote debugging, install the [App Service extension](#). Follow the instructions on the extension page and sign in to Azure in Visual Studio Code.

In the Azure explorer, find the app you want to debug, right-click it and select **Start Remote Debugging**. Click **Yes** to enable it for your app. App Service starts a tunnel proxy for you and attaches the debugger. You can then make requests to the app and see the debugger pausing at break points.

Once finished with debugging, stop the debugger by selecting **Disconnect**. When prompted, you should click

Yes to disable remote debugging. To disable it later, right-click your app again in the Azure explorer and select **Disable Remote Debugging**.

## Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them using the standard Node.js pattern. For example, to access an app setting called `NODE_ENV`, use the following code:

```
process.env.NODE_ENV
```

## Run Grunt/Bower/Gulp

By default, App Service build automation runs `npm install --production` when it recognizes a Node.js app is deployed through Git, or through Zip deployment [with build automation enabled](#). If your app requires any of the popular automation tools, such as Grunt, Bower, or Gulp, you need to supply a [custom deployment script](#) to run it.

To enable your repository to run these tools, you need to add them to the dependencies in `package.json`. For example:

```
"dependencies": {
 "bower": "^1.7.9",
 "grunt": "^1.0.1",
 "gulp": "^3.9.1",
 ...
}
```

From a local terminal window, change directory to your repository root and run the following commands:

```
npm install kuduscript -g
kuduscript --node --scriptType bash --suppressPrompt
```

Your repository root now has two additional files: `.deployment` and `deploy.sh`.

Open `deploy.sh` and find the `Deployment` section, which looks like this:

```
#####
Deployment

```

This section ends with running `npm install --production`. Add the code section you need to run the required tool *at the end of* the `Deployment` section:

- [Bower](#)
- [Gulp](#)
- [Grunt](#)

See an [example in the MEAN.js sample](#), where the deployment script also runs a custom `npm install` command.

### Bower

This snippet runs `bower install`.

```
if [-e "$DEPLOYMENT_TARGET/bower.json"]; then
 cd "$DEPLOYMENT_TARGET"
 eval ./node_modules/.bin/bower install
 exitWithMessageOnError "bower failed"
 cd - > /dev/null
fi
```

## Gulp

This snippet runs `gulp imagemin`.

```
if [-e "$DEPLOYMENT_TARGET/gulpfile.js"]; then
 cd "$DEPLOYMENT_TARGET"
 eval ./node_modules/.bin/gulp imagemin
 exitWithMessageOnError "gulp failed"
 cd - > /dev/null
fi
```

## Grunt

This snippet runs `grunt`.

```
if [-e "$DEPLOYMENT_TARGET/Gruntfile.js"]; then
 cd "$DEPLOYMENT_TARGET"
 eval ./node_modules/.bin/grunt
 exitWithMessageOnError "Grunt failed"
 cd - > /dev/null
fi
```

## Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

Popular web frameworks let you access the `X-Forwarded-*` information in your standard app pattern. In [Express](#), you can use [trust proxies](#). For example:

```
app.set('trust proxy', 1)
...
if (req.secure) {
 // Do something when HTTPS is used
}
```

## Access diagnostic logs

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging
filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

**NOTE**

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

```
az webapp log config --name <app-name> --resource-group <resource-group-name> --docker-container-logging filesystem
```

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

Once container logging is turned on, run the following command to see the log stream:

```
az webapp log tail --name <app-name> --resource-group <resource-group-name>
```

If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type `Ctrl+C`.

You can also inspect the log files in a browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

## Monitor with Application Insights

Application Insights allows you to monitor your application's performance, exceptions, and usage without making any code changes. To attach the App Insights agent, go to your web app in the Portal and select **Application Insights** under **Settings**, then select **Turn on Application Insights**. Next, select an existing App Insights resource or create a new one. Finally, select **Apply** at the bottom. To instrument your web app using PowerShell, please see [these instructions](#)

This agent will monitor your server-side Node.js application. To monitor your client-side JavaScript, [add the JavaScript SDK to your project](#).

For more information, see the [Application Insights extension release notes](#).

## Troubleshooting

When a working Node.js app behaves differently in App Service or has errors, try the following:

- [Access the log stream](#).
- Test the app locally in production mode. App Service runs your Node.js apps in production mode, so you need to make sure that your project works as expected in production mode locally. For example:
  - Depending on your `package.json`, different packages may be installed for production mode (`dependencies` vs. `devDependencies`).
  - Certain web frameworks may deploy static files differently in production mode.
  - Certain web frameworks may use custom startup scripts when running in production mode.

- Run your app in App Service in development mode. For example, in [MEAN.js](#), you can set your app to development mode in runtime by setting the `NODE_ENV` app setting.

#### You do not have permission to view this directory or page

After deploying your Node.js code to a native Windows app in App Service, you may see the message

You do not have permission to view this directory or page.

This is most likely because you don't have a `web.config` file (see the [template](#) and an [example](#)).

If you deploy your files by using Git, or by using ZIP deployment [with build automation enabled](#), the deployment engine generates a `web.config` in the web root of your app (`%HOME%\site\wwwroot`) automatically if one of the following conditions is true:

- Your project root has a `package.json` that defines a `start` script that contains the path of a JavaScript file.
- Your project root has either a `server.js` or an `app.js`.

The generated `web.config` is tailored to the detected start script. For other deployment methods, add this `web.config` manually. Make sure the file is formatted properly.

If you use [ZIP deployment](#) (through Visual Studio Code, for example), be sure to [enable build automation](#) because it's not enabled by default. `az webapp up` uses ZIP deployment with build automation enabled.

## robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415
"-_-_-_-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

## Next steps

[Tutorial: Node.js app with MongoDB](#)

[App Service Linux FAQ](#)

Or, see additional resources:

[Environment variables and app settings reference](#)

# Configure a PHP app for Azure App Service

11/2/2021 • 12 minutes to read • [Edit Online](#)

This guide shows you how to configure your PHP web apps, mobile back ends, and API apps in Azure App Service.

This guide provides key concepts and instructions for PHP developers who deploy apps to App Service. If you've never used Azure App Service, follow the [PHP quickstart](#) and [PHP with MySQL tutorial](#) first.

## Show PHP version

To show the current PHP version, run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query phpVersion
```

### NOTE

To address a development slot, include the parameter `--slot` followed by the name of the slot.

To show all supported PHP versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes | grep php
```

To show the current PHP version, run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query linuxFxVersion
```

### NOTE

To address a development slot, include the parameter `--slot` followed by the name of the slot.

To show all supported PHP versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes --linux | grep PHP
```

## Set PHP version

Run the following command in the [Cloud Shell](#) to set the PHP version to 7.4:

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --php-version 7.4
```

Run the following command in the [Cloud Shell](#) to set the PHP version to 7.2:

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "PHP|7.2"
```

## Run Composer

If you want App Service to run [Composer](#) at deployment time, the easiest way is to include the Composer in your repository.

From a local terminal window, change directory to your repository root, and follow the instructions at [download Composer](#) to download `composer.phar` to the directory root.

Run the following commands (you need [npm](#) installed):

```
npm install kuduscript -g
kuduscript --node --scriptType bash --suppressPrompt
```

Your repository root now has two additional files: `.deployment` and `deploy.sh`.

Open `deploy.sh` and find the `Deployment` section, which looks like this:

```
#####
Deployment

```

Add the code section you need to run the required tool *at the end* of the `Deployment` section:

```
4. Use composer
echo "$DEPLOYMENT_TARGET"
if [-e "$DEPLOYMENT_TARGET/composer.json"]; then
 echo "Found composer.json"
 pushd "$DEPLOYMENT_TARGET"
 php composer.phar install $COMPOSER_ARGS
 exitWithMessageOnError "Composer install failed"
 popd
fi
```

Commit all your changes and deploy your code using Git, or Zip deploy [with build automation enabled](#). Composer should now be running as part of deployment automation.

## Run Grunt/Bower/Gulp

If you want App Service to run popular automation tools at deployment time, such as Grunt, Bower, or Gulp, you need to supply a [custom deployment script](#). App Service runs this script when you deploy with Git, or with [Zip deployment](#) with [with build automation enabled](#).

To enable your repository to run these tools, you need to add them to the dependencies in `package.json`. For example:

```
"dependencies": {
 "bower": "^1.7.9",
 "grunt": "^1.0.1",
 "gulp": "^3.9.1",
 ...
}
```

From a local terminal window, change directory to your repository root and run the following commands (you need [npm](#) installed):

```
npm install kuduscript -g
kuduscript --node --scriptType bash --suppressPrompt
```

Your repository root now has two additional files: `.deployment` and `deploy.sh`.

Open `deploy.sh` and find the `Deployment` section, which looks like this:

```
#####
Deployment

```

This section ends with running `npm install --production`. Add the code section you need to run the required tool *at the end* of the `Deployment` section:

- [Bower](#)
- [Gulp](#)
- [Grunt](#)

See an [example in the MEAN.js sample](#), where the deployment script also runs a custom `npm install` command.

## Bower

This snippet runs `bower install`.

```
if [-e "$DEPLOYMENT_TARGET/bower.json"]; then
 cd "$DEPLOYMENT_TARGET"
 eval ./node_modules/.bin/bower install
 exitWithErrorOnFailure "bower failed"
 cd - > /dev/null
fi
```

## Gulp

This snippet runs `gulp imagemin`.

```
if [-e "$DEPLOYMENT_TARGET/gulpfile.js"]; then
 cd "$DEPLOYMENT_TARGET"
 eval ./node_modules/.bin/gulp imagemin
 exitWithErrorOnFailure "gulp failed"
 cd - > /dev/null
fi
```

## Grunt

This snippet runs `grunt`.

```
if [-e "$DEPLOYMENT_TARGET/Gruntfile.js"]; then
 cd "$DEPLOYMENT_TARGET"
 eval ./node_modules/.bin/grunt
 exitWithErrorOnFailure "Grunt failed"
 cd - > /dev/null
fi
```

# Customize build automation

If you deploy your app using Git, or using zip packages [with build automation enabled](#), the App Service build automation steps through the following sequence:

1. Run custom script if specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `php composer.phar install`.
3. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

`PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` are environment variables that are empty by default. To run pre-build commands, define `PRE_BUILD_COMMAND`. To run post-build commands, define `POST_BUILD_COMMAND`.

The following example specifies the two variables to a series of commands, separated by commas.

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
PRE_BUILD_COMMAND="echo foo, scripts/prebuild.sh"
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
POST_BUILD_COMMAND="echo foo, scripts/postbuild.sh"
```

For additional environment variables to customize build automation, see [Oryx configuration](#).

For more information on how App Service runs and builds PHP apps in Linux, see [Oryx documentation: How PHP apps are detected and built](#).

## Customize start-up

By default, the built-in PHP container runs the Apache server. At start-up, it runs `apache2ctl -D FOREGROUND`. If you like, you can run a different command at start-up, by running the following command in the [Cloud Shell](#):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<custom-command>"
```

## Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them using the standard `getenv()` pattern. For example, to access an app setting called `DB_HOST`, use the following code:

```
getenv("DB_HOST")
```

## Change site root

The web framework of your choice may use a subdirectory as the site root. For example, [Laravel](#), uses the `public`/subdirectory as the site root.

To customize the site root, set the virtual application path for the app by using the `az resource update` command. The following example sets the site root to the `public`/subdirectory in your repository.

```
az resource update --name web --resource-group <group-name> --namespace Microsoft.Web --resource-type config
--parent sites/<app-name> --set properties.virtualApplications[0].physicalPath="site\wwwroot\public" --api-version 2015-06-01
```

By default, Azure App Service points the root virtual application path (`/`) to the root directory of the deployed application files (`sites\wwwroot`).

The web framework of your choice may use a subdirectory as the site root. For example, [Laravel](#), uses the

`public/` subdirectory as the site root.

The default PHP image for App Service uses Apache, and it doesn't let you customize the site root for your app. To work around this limitation, add an `.htaccess` file to your repository root with the following content:

```
<IfModule mod_rewrite.c>
 RewriteEngine on
 RewriteCond %{REQUEST_URI} ^(.*)
 RewriteRule ^(.*)$ /public/$1 [NC,L,QSA]
</IfModule>
```

If you would rather not use `.htaccess` rewrite, you can deploy your Laravel application with a [custom Docker image](#) instead.

## Detect HTTPS session

In App Service, [TLS/SSL termination](#) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

```
if (isset($_SERVER['HTTP_X_FORWARDED_PROTO']) && $_SERVER['HTTP_X_FORWARDED_PROTO'] === 'https') {
 // Do something when HTTPS is used
}
```

Popular web frameworks let you access the `X-Forwarded-*` information in your standard app pattern. In [CodeIgniter](#), the `is_https()` checks the value of `X_FORWARDED_PROTO` by default.

## Customize php.ini settings

If you need to make changes to your PHP installation, you can change any of the [php.ini directives](#) by following these steps.

### NOTE

The best way to see the PHP version and the current `php.ini` configuration is to call `phpinfo()` in your app.

### Customize-non-PHP\_INI\_SYSTEM directives

To customize `PHP_INI_USER`, `PHP_INI_PERDIR`, and `PHP_INI_ALL` directives (see [php.ini directives](#)), add a `.user.ini` file to the root directory of your app.

Add configuration settings to the `.user.ini` file using the same syntax you would use in a `php.ini` file. For example, if you wanted to turn on the `display_errors` setting and set `upload_max_filesize` setting to 10M, your `.user.ini` file would contain this text:

```
; Example Settings
display_errors=On
upload_max_filesize=10M

; Write errors to d:\home\LogFiles\php_errors.log
; log_errors=On
```

Redeploy your app with the changes and restart it.

As an alternative to using a `.user.ini` file, you can use `ini_set()` in your app to customize these non-`PHP_INI_SYSTEM` directives.

To customize PHP\_INI\_USER, PHP\_INI\_PERDIR, and PHP\_INI\_ALL directives (see [php.ini directives](#)), add an `.htaccess` file to the root directory of your app.

In the `.htaccess` file, add the directives using the `php_value <directive-name> <value>` syntax. For example:

```
php_value upload_max_filesize 1000M
php_value post_max_size 2000M
php_value memory_limit 3000M
php_value max_execution_time 180
php_value max_input_time 180
php_value display_errors On
php_value upload_max_filesize 10M
```

Redeploy your app with the changes and restart it. If you deploy it with Kudu (for example, using [Git](#)), it's automatically restarted after deployment.

As an alternative to using `.htaccess`, you can use [ini\\_set\(\)](#) in your app to customize these non-PHP\_INI\_SYSTEM directives.

### Customize PHP\_INI\_SYSTEM directives

To customize PHP\_INI\_SYSTEM directives (see [php.ini directives](#)), you can't use the `.htaccess` approach. App Service provides a separate mechanism using the `PHP_INI_SCAN_DIR` app setting.

First, run the following command in the [Cloud Shell](#) to add an app setting called `PHP_INI_SCAN_DIR`:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
PHP_INI_SCAN_DIR="d:\home\site\ini"
```

Navigate to the Kudu console (<https://<app-name>.scm.azurewebsites.net/DebugConsole>) and navigate to `d:\home\site`.

Create a directory in `d:\home\site` called `ini`, then create an `.ini` file in the `d:\home\site\ini` directory (for example, `settings.ini`) with the directives you want to customize. Use the same syntax you would use in a `php.ini` file.

For example, to change the value of `expose_php` run the following commands:

```
cd /home/site
mkdir ini
echo "expose_php = Off" >> ini/setting.ini
```

For the changes to take effect, restart the app.

To customize PHP\_INI\_SYSTEM directives (see [php.ini directives](#)), you can't use the `.htaccess` approach. App Service provides a separate mechanism using the `PHP_INI_SCAN_DIR` app setting.

First, run the following command in the [Cloud Shell](#) to add an app setting called `PHP_INI_SCAN_DIR`:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
PHP_INI_SCAN_DIR="/usr/local/etc/php/conf.d:/home/site/ini"
```

`/usr/local/etc/php/conf.d` is the default directory where `php.ini` exists. `/home/site/ini` is the custom directory in which you'll add a custom `.ini` file. You separate the values with a `:`.

Navigate to the web SSH session with your Linux container (

<https://<app-name>.scm.azurewebsites.net/webssh/host> ).

Create a directory in `/home/site` called `ini`, then create an `.ini` file in the `/home/site/ini` directory (for example, `settings.ini`) with the directives you want to customize. Use the same syntax you would use in a `php.ini` file.

**TIP**

In the built-in Linux containers in App Service, `/home` is used as persisted shared storage.

For example, to change the value of `expose_php` run the following commands:

```
cd /home/site
mkdir ini
echo "expose_php = Off" >> ini/setting.ini
```

For the changes to take effect, restart the app.

## Enable PHP extensions

The built-in PHP installations contain the most commonly used extensions. You can enable additional extensions in the same way that you [customize php.ini directives](#).

**NOTE**

The best way to see the PHP version and the current `php.ini` configuration is to call `phpinfo()` in your app.

To enable additional extensions, by following these steps:

Add a `bin` directory to the root directory of your app and put the `.dll` extension files in it (for example, `mongodb.dll`). Make sure that the extensions are compatible with the PHP version in Azure and are VC9 and non-thread-safe (nts) compatible.

Deploy your changes.

Follow the steps in [Customize PHP\\_INI\\_SYSTEM directives](#), add the extensions into the custom `.ini` file with the `extension` or `zend_extension` directives.

```
extension=d:\home\site\wwwroot\bin\mongodb.dll
zend_extension=d:\home\site\wwwroot\bin\xdebug.dll
```

For the changes to take effect, restart the app.

The built-in PHP installations contain the most commonly used extensions. You can enable additional extensions in the same way that you [customize php.ini directives](#).

**NOTE**

The best way to see the PHP version and the current `php.ini` configuration is to call `phpinfo()` in your app.

To enable additional extensions, by following these steps:

Add a `bin` directory to the root directory of your app and put the `.so` extension files in it (for example, `mongodb.so`). Make sure that the extensions are compatible with the PHP version in Azure and are VC9 and

non-thread-safe (nts) compatible.

Deploy your changes.

Follow the steps in [Customize PHP\\_INI\\_SYSTEM directives](#), add the extensions into the custom *.ini* file with the `extension` or `zend_extension` directives.

```
extension=/home/site/wwwroot/bin/mongodb.so
zend_extension=/home/site/wwwroot/bin/xdebug.so
```

For the changes to take effect, restart the app.

## Access diagnostic logs

Use the standard `error_log()` utility to make your diagnostic logs to show up in Azure App Service.

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging
filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

### NOTE

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

```
az webapp log config --name <app-name> --resource-group <resource-group-name> --docker-container-logging
filesystem
```

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

Once container logging is turned on, run the following command to see the log stream:

```
az webapp log tail --name <app-name> --resource-group <resource-group-name>
```

If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type `Ctrl+C`.

You can also inspect the log files in a browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

## Troubleshooting

When a working PHP app behaves differently in App Service or has errors, try the following:

- [Access the log stream](#).
- Test the app locally in production mode. App Service runs your app in production mode, so you need to make sure that your project works as expected in production mode locally. For example:
  - Depending on your *composer.json*, different packages may be installed for production mode (`require` vs. `require-dev`).
  - Certain web frameworks may deploy static files differently in production mode.
  - Certain web frameworks may use custom startup scripts when running in production mode.
- Run your app in App Service in debug mode. For example, in [Laravel](#), you can configure your app to output debug messages in production by [setting the `APP\_DEBUG` app setting to `true`](#).

## robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415
"-"
"-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

## Next steps

[Tutorial: PHP app with MySQL](#)

[App Service Linux FAQ](#)

Or, see additional resources:

[Environment variables and app settings reference](#)

# Configure a Linux Python app for Azure App Service

11/2/2021 • 21 minutes to read • [Edit Online](#)

This article describes how [Azure App Service](#) runs Python apps, how you can migrate existing apps to Azure, and how you can customize the behavior of App Service when needed. Python apps must be deployed with all the required [pip](#) modules.

The App Service deployment engine automatically activates a virtual environment and runs

`pip install -r requirements.txt` for you when you deploy a [Git repository](#), or a [zip package with build automation enabled](#).

This guide provides key concepts and instructions for Python developers who use a built-in Linux container in App Service. If you've never used Azure App Service, first follow the [Python quickstart](#) and [Python with PostgreSQL tutorial](#).

You can use either the [Azure portal](#) or the Azure CLI for configuration:

- [Azure portal](#), use the app's **Settings > Configuration** page as described on [Configure an App Service app in the Azure portal](#).
- [Azure CLI](#): you have two options.
  - Run commands in the [Azure Cloud Shell](#).
  - Run commands locally by installing the latest version of the [Azure CLI](#), then sign in to Azure using `az login`.

## NOTE

Linux is currently the recommended option for running Python apps in App Service. For information on the Windows option, see [Python on the Windows flavor of App Service](#).

## Configure Python version

- [Azure portal](#): use the **General settings** tab on the **Configuration** page as described on [Configure general settings](#) for Linux containers.
- [Azure CLI](#):
  - Show the current Python version with `az webapp config show`:

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query
linuxFxVersion
```

Replace `<resource-group-name>` and `<app-name>` with the names appropriate for your web app.

- Set the Python version with `az webapp config set`

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-
version "PYTHON|3.7"
```

- Show all Python versions that are supported in Azure App Service with [az webapp list-runtimes](#):

```
az webapp list-runtimes --linux | grep PYTHON
```

You can run an unsupported version of Python by building your own container image instead. For more information, see [use a custom Docker image](#).

## Customize build automation

App Service's build system, called Oryx, performs the following steps when you deploy your app if the app setting `SCM_DO_BUILD_DURING_DEPLOYMENT` is set to `1`:

1. Run a custom pre-build script if specified by the `PRE_BUILD_COMMAND` setting. (The script can itself run other Python and Nodejs scripts, pip and npm commands, and Node-based tools like yarn, for example, `yarn install` and `yarn build`.)
2. Run `pip install -r requirements.txt`. The `requirements.txt` file must be present in the project's root folder. Otherwise, the build process reports the error: "Could not find setup.py or requirements.txt; Not running pip install."
3. If `manage.py` is found in the root of the repository (indicating a Django app), run `manage.py collectstatic`. However, if the `DISABLE_COLLECTSTATIC` setting is `true`, this step is skipped.
4. Run custom post-build script if specified by the `POST_BUILD_COMMAND` setting. (Again, the script can run other Python and Nodejs scripts, pip and npm commands, and Node-based tools.)

By default, the `PRE_BUILD_COMMAND`, `POST_BUILD_COMMAND`, and `DISABLE_COLLECTSTATIC` settings are empty.

- To disable running collectstatic when building Django apps, set the `DISABLE_COLLECTSTATIC` setting to true.
- To run pre-build commands, set the `PRE_BUILD_COMMAND` setting to contain either a command, such as `echo Pre-build command`, or a path to a script file relative to your project root, such as `scripts/prebuild.sh`. All commands must use relative paths to the project root folder.
- To run post-build commands, set the `POST_BUILD_COMMAND` setting to contain either a command, such as `echo Post-build command`, or a path to a script file relative to your project root, such as `scripts/postbuild.sh`. All commands must use relative paths to the project root folder.

For additional settings that customize build automation, see [Oryx configuration](#).

To access the build and deployment logs, see [Access deployment logs](#).

For more information on how App Service runs and builds Python apps in Linux, see [How Oryx detects and builds Python apps](#).

### NOTE

The `PRE_BUILD_SCRIPT_PATH` and `POST_BUILD_SCRIPT_PATH` settings are identical to `PRE_BUILD_COMMAND` and `POST_BUILD_COMMAND` and are supported for legacy purposes.

A setting named `SCM_DO_BUILD_DURING_DEPLOYMENT`, if it contains `true` or 1, triggers an Oryx build happens during deployment. The setting is true when deploying using git, the Azure CLI command `az webapp up`, and Visual Studio Code.

## NOTE

Always use relative paths in all pre- and post-build scripts because the build container in which Oryx runs is different from the runtime container in which the app runs. Never rely on the exact placement of your app project folder within the container (for example, that it's placed under `site/wwwroot`).

# Migrate existing applications to Azure

Existing web applications can be redeployed to Azure as follows:

1. **Source repository:** Maintain your source code in a suitable repository like GitHub, which enables you to set up continuous deployment later in this process.
  - a. Your `requirements.txt` file must be at the root of your repository for App Service to automatically install the necessary packages.
2. **Database:** If your app depends on a database, provision the necessary resources on Azure as well. See [Tutorial: Deploy a Django web app with PostgreSQL - create a database](#) for an example.
3. **App service resources:** Create a resource group, App Service Plan, and App Service web app to host your application. You can most easily do this by doing an initial deployment of your code through the Azure CLI command `az webapp up`, as shown on [Tutorial: Deploy a Django web app with PostgreSQL - deploy the code](#). Replace the names of the resource group, App Service Plan, and the web app to be more suitable for your application.
4. **Environment variables:** If your application requires any environment variables, create equivalent [App Service application settings](#). These App Service settings appear to your code as environment variables, as described on [Access environment variables](#).
  - Database connections, for example, are often managed through such settings, as shown in [Tutorial: Deploy a Django web app with PostgreSQL - configure variables to connect the database](#).
  - See [Production settings for Django apps](#) for specific settings for typical Django apps.
5. **App startup:** Review the section, [Container startup process](#) later in this article to understand how App Service attempts to run your app. App Service uses the Gunicorn web server by default, which must be able to find your app object or `wsgi.py` folder. If needed, you can [Customize the startup command](#).
6. **Continuous deployment:** Set up continuous deployment, as described on [Continuous deployment to Azure App Service](#) if using Azure Pipelines or Kudu deployment, or [Deploy to App Service using GitHub Actions](#) if using GitHub actions.
7. **Custom actions:** To perform actions within the App Service container that hosts your app, such as Django database migrations, you can [connect to the container through SSH](#). For an example of running Django database migrations, see [Tutorial: Deploy a Django web app with PostgreSQL - run database migrations](#).
  - When using continuous deployment, you can perform those actions using post-build commands as described earlier under [Customize build automation](#).

With these steps completed, you should be able to commit changes to your source repository and have those updates automatically deployed to App Service.

## Production settings for Django apps

For a production environment like Azure App Service, Django apps should follow Django's [Deployment checklist](#) ([djangoproject.com](http://djangoproject.com)).

The following table describes the production settings that are relevant to Azure. These settings are defined in the

app's `setting.py` file.

DJANGO SETTING	INSTRUCTIONS FOR AZURE
<code>SECRET_KEY</code>	Store the value in an App Service setting as described on <a href="#">Access app settings as environment variables</a> . You can alternately <a href="#">store the value as a "secret" in Azure Key Vault</a> .
<code>DEBUG</code>	Create a <code>DEBUG</code> setting on App Service with the value 0 (false), then load the value as an environment variable. In your development environment, create a <code>DEBUG</code> environment variable with the value 1 (true).
<code>ALLOWED_HOSTS</code>	In production, Django requires that you include app's URL in the <code>ALLOWED_HOSTS</code> array of <code>settings.py</code> . You can retrieve this URL at runtime with the code, <code>os.environ['WEBSITE_HOSTNAME']</code> . App Service automatically sets the <code>WEBSITE_HOSTNAME</code> environment variable to the app's URL.
<code>DATABASES</code>	Define settings in App Service for the database connection and load them as environment variables to populate the <code>DATABASES</code> dictionary. You can alternately store the values (especially the username and password) as <a href="#">Azure Key Vault secrets</a> .

## Serve static files for Django apps

If your Django web app includes static front-end files, first follow the instructions on [Managing static files](#) in the Django documentation.

For App Service, you then make the following modifications:

1. Consider using environment variables (for local development) and App Settings (when deploying to the cloud) to dynamically set the Django `STATIC_URL` and `STATIC_ROOT` variables. For example:

```
STATIC_URL = os.environ.get("DJANGO_STATIC_URL", "/static/")
STATIC_ROOT = os.environ.get("DJANGO_STATIC_ROOT", "./static/")
```

`DJANGO_STATIC_URL` and `DJANGO_STATIC_ROOT` can be changed as necessary for your local and cloud environments. For example, if the build process for your static files places them in a folder named `django-static`, then you can set `DJANGO_STATIC_URL` to `/django-static/` to avoid using the default.

2. If you have a pre-build script that generates static files in a different folder, include that folder in the Django `STATICFILES_DIRS` variable so that Django's `collectstatic` process finds them. For example, if you run `yarn build` in your front-end folder, and yarn generates a `build/static` folder containing static files, then include that folder as follows:

```
FRONTEND_DIR = "path-to-frontend-folder"
STATICFILES_DIRS = [os.path.join(FRONTEND_DIR, 'build', 'static')]
```

Here, `FRONTEND_DIR`, to build a path to where a build tool like yarn is run. You can again use an environment variable and App Setting as desired.

3. Add `whitenoise` to your `requirements.txt` file. [Whitenoise](#) (`whitenoise.evans.io`) is a Python package that makes it simple for a production Django app to serve its own static files. Whitenoise specifically serves

those files that are found in the folder specified by the Django `STATIC_ROOT` variable.

4. In your `settings.py` file, add the following line for Whitenoise:

```
STATICFILES_STORAGE = ('whitenoise.storage.CompressedManifestStaticFilesStorage')
```

5. Also modify the `MIDDLEWARE` and `INSTALLED_APPS` lists to include Whitenoise:

```
MIDDLEWARE = [
 'django.middleware.security.SecurityMiddleware',
 # Add whitenoise middleware after the security middleware
 'whitenoise.middleware.WhiteNoiseMiddleware',
 # Other values follow
]

INSTALLED_APPS = [
 "whitenoise.runserver_nostatic",
 # Other values follow
]
```

## Container characteristics

When deployed to App Service, Python apps run within a Linux Docker container that's defined in the [App Service Python GitHub repository](#). You can find the image configurations inside the version-specific directories.

This container has the following characteristics:

- Apps are run using the [Gunicorn WSGI HTTP Server](#), using the additional arguments `--bind=0.0.0.0 --timeout 600`.
  - You can provide configuration settings for Gunicorn through a `gunicorn.conf.py` file in the project root, as described on [Gunicorn configuration overview](#) (docs.gunicorn.org). You can alternately [customize the startup command](#).
  - To protect your web app from accidental or deliberate DDOS attacks, Gunicorn is run behind an Nginx reverse proxy as described on [Deploying Gunicorn](#) (docs.gunicorn.org).
- By default, the base container image includes only the Flask web framework, but the container supports other frameworks that are WSGI-compliant and compatible with Python 3.6+, such as Django.
- To install additional packages, such as Django, create a `requirements.txt` file in the root of your project that specifies your direct dependencies. App Service then installs those dependencies automatically when you deploy your project.

The `requirements.txt` file must be in the project root for dependencies to be installed. Otherwise, the build process reports the error: "Could not find setup.py or requirements.txt; Not running pip install." If you encounter this error, check the location of your requirements file.

- App Service automatically defines an environment variable named `WEBSITE_HOSTNAME` with the web app's URL, such as `msdocs-hello-world.azurewebsites.net`. It also defines `WEBSITE_SITE_NAME` with the name of your app, such as `msdocs-hello-world`.
- npm and Node.js are installed in the container so you can run Node-based build tools, such as yarn.

## Container startup process

During startup, the App Service on Linux container runs the following steps:

1. Use a [custom startup command](#), if provided.
2. Check for the existence of a [Django app](#), and launch Gunicorn for it if detected.
3. Check for the existence of a [Flask app](#), and launch Gunicorn for it if detected.
4. If no other app is found, start a default app that's built into the container.

The following sections provide additional details for each option.

### Django app

For Django apps, App Service looks for a file named `wsgi.py` within your app code, and then runs Gunicorn using the following command:

```
<module> is the name of the folder that contains wsgi.py
gunicorn --bind=0.0.0.0 --timeout 600 <module>.wsgi
```

If you want more specific control over the startup command, use a [custom startup command](#), replace `<module>` with the name of folder that contains `wsgi.py`, and add a `--chdir` argument if that module is not in the project root. For example, if your `wsgi.py` is located under `knboard/backend/config` from your project root, use the arguments `--chdir knboard/backend config.wsgi`.

To enable production logging, add the `--access-logfile` and `--error-logfile` parameters as shown in the examples for [custom startup commands](#).

### Flask app

For Flask, App Service looks for a file named `application.py` or `app.py` and starts Gunicorn as follows:

```
If application.py
gunicorn --bind=0.0.0.0 --timeout 600 application:app

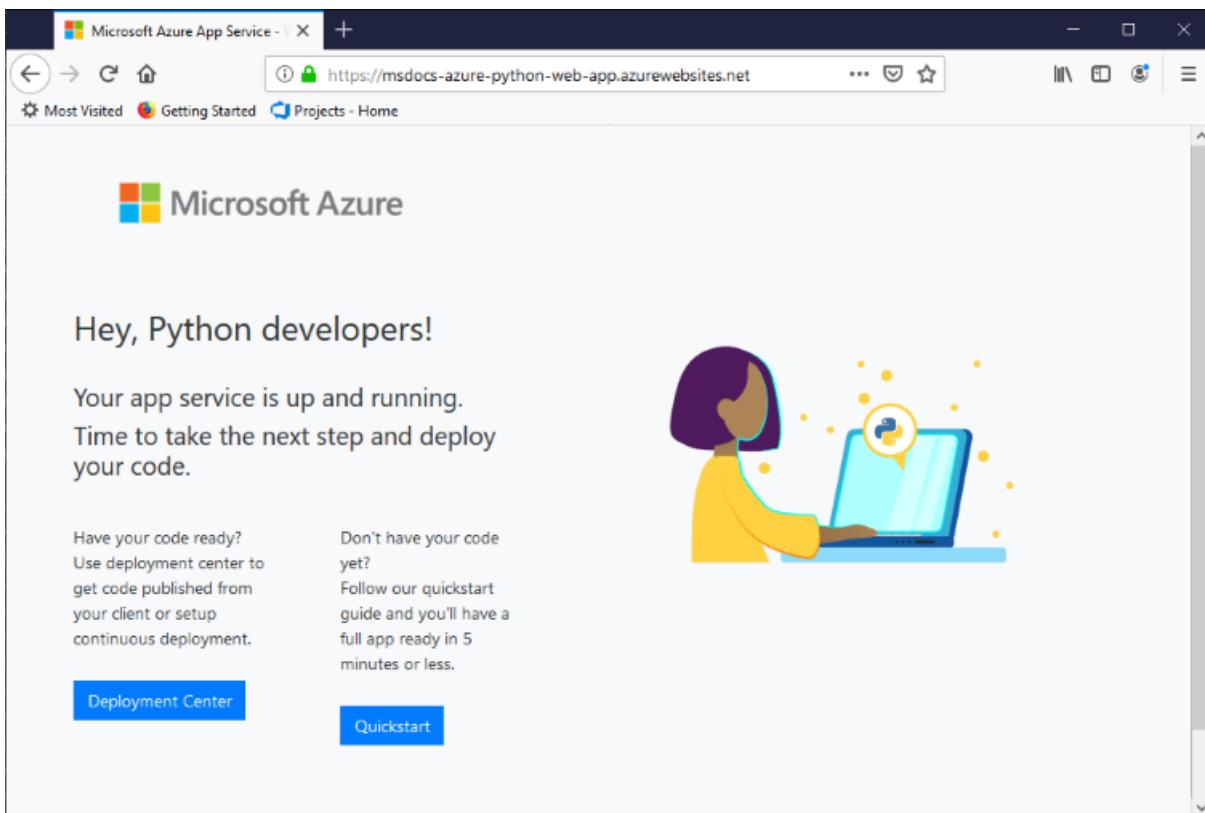
If app.py
gunicorn --bind=0.0.0.0 --timeout 600 app:app
```

If your main app module is contained in a different file, use a different name for the app object, or you want to provide additional arguments to Gunicorn, use a [custom startup command](#).

### Default behavior

If the App Service doesn't find a custom command, a Django app, or a Flask app, then it runs a default read-only app, located in the `opt/defaultsite` folder and shown in the following image.

If you deployed code and still see the default app, see [Troubleshooting - App doesn't appear](#).



Again, if you expect to see a deployed app instead of the default app, see [Troubleshooting - App doesn't appear](#).

## Customize startup command

As noted earlier in this article, you can provide configuration settings for Gunicorn through a `gunicorn.conf.py` file in the project root, as described on [Gunicorn configuration overview](#).

If such configuration is not sufficient, you can control the container's startup behavior by providing either a custom startup command or multiple commands in a startup command file. A startup command file can use whatever name you choose, such as `startup.sh`, `startup.cmd`, `startup.txt`, and so on.

All commands must use relative paths to the project root folder.

To specify a startup command or command file:

- **Azure portal:** select the app's [Configuration](#) page, then select [General settings](#). In the **Startup Command** field, place either the full text of your startup command or the name of your startup command file. Then select **Save** to apply the changes. See [Configure general settings](#) for Linux containers.
- **Azure CLI:** use the `az webapp config set` command with the `--startup-file` parameter to set the startup command or file:

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<custom-command>"
```

Replace `<custom-command>` with either the full text of your startup command or the name of your startup command file.

App Service ignores any errors that occur when processing a custom startup command or file, then continues its startup process by looking for Django and Flask apps. If you don't see the behavior you expect, check that your startup command or file is error-free and that a startup command file is deployed to App Service along with your app code. You can also check the [Diagnostic logs](#) for additional information. Also check the app's

[Diagnose and solve problems](#) page on the [Azure portal](#).

## Example startup commands

- **Added Gunicorn arguments:** The following example adds the `--workers=4` to a Gunicorn command line for starting a Django app:

```
<module-path> is the relative path to the folder that contains the module
that contains wsgi.py; <module> is the name of the folder containing wsgi.py.
gunicorn --bind=0.0.0.0 --timeout 600 --workers=4 --chdir <module_path> <module>.wsgi
```

For more information, see [Running Gunicorn](#) ([docs.gunicorn.org](https://docs.gunicorn.org)).

- **Enable production logging for Django:** Add the `--access-logfile '-'` and `--error-logfile '-'` arguments to the command line:

```
'-' for the log files means stdout for --access-logfile and stderr for --error-logfile.
gunicorn --bind=0.0.0.0 --timeout 600 --workers=4 --chdir <module_path> <module>.wsgi --access-
logfile '-' --error-logfile '-'
```

These logs will appear in the [App Service log stream](#).

For more information, see [Gunicorn logging](#) ([docs.gunicorn.org](https://docs.gunicorn.org)).

- **Custom Flask main module:** by default, App Service assumes that a Flask app's main module is `application.py` or `app.py`. If your main module uses a different name, then you must customize the startup command. For example, if you have a Flask app whose main module is `hello.py` and the Flask app object in that file is named `myapp`, then the command is as follows:

```
gunicorn --bind=0.0.0.0 --timeout 600 hello:myapp
```

If your main module is in a subfolder, such as `website`, specify that folder with the `--chdir` argument:

```
gunicorn --bind=0.0.0.0 --timeout 600 --chdir website hello:myapp
```

- **Use a non-Gunicorn server:** To use a different web server, such as [aiohttp](#), use the appropriate command as the startup command or in the startup command file:

```
python3.7 -m aiohttp.web -H localhost -P 8080 package.module:init_func
```

## Access app settings as environment variables

App settings are values stored in the cloud specifically for your app as described on [Configure app settings](#). These settings are available to your app code as environment variables and accessed using the standard `os.environ` pattern.

For example, if you've created app setting called `DATABASE_SERVER`, the following code retrieves that setting's value:

```
db_server = os.environ['DATABASE_SERVER']
```

## Detect HTTPS session

In App Service, [TLS/SSL termination](#) (wikipedia.org) happens at the network load balancers, so all HTTPS requests reach your app as unencrypted HTTP requests. If your app logic needs to check if the user requests are encrypted or not, inspect the `X-Forwarded-Proto` header.

```
if 'X-Forwarded-Proto' in request.headers and request.headers['X-Forwarded-Proto'] == 'https':
 # Do something when HTTPS is used
```

Popular web frameworks let you access the `x-Forwarded-*` information in your standard app pattern. In [Codeigniter](#), the `is_https()` checks the value of `X_FORWARDED_PROTO` by default.

## Access diagnostic logs

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

```
az webapp log config --name <app-name> --resource-group <resource-group-name> --docker-container-logging
filesystem
```

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

Once container logging is turned on, run the following command to see the log stream:

```
az webapp log tail --name <app-name> --resource-group <resource-group-name>
```

If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type **Ctrl+C**.

You can also inspect the log files in a browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To access logs through the Azure portal, select **Monitoring > Log stream** on the left side menu for your app.

## Access deployment logs

When you deploy your code, App Service performs the build process described earlier in the section [Customize build automation](#). Because the build runs in its own container, build logs are stored separately from the app's diagnostic logs.

Use the following steps to access the deployment logs:

1. On the Azure portal for your web app, select **Deployment > Deployment Center (Preview)** on the left menu.
2. On the **Logs** tab, select the **Commit ID** for the most recent commit.
3. On the **Log details** page that appears, select the **Show Logs...** link that appears next to "Running oryx build...".

Build issues such as incorrect dependencies in `requirements.txt` and errors in pre- or post-build scripts will appear in these logs. Errors also appear if your requirements file is not exactly named `requirements.txt` or does not appear in the root folder of your project.

## Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace <app-name> with your app name:

```
https://<app-name>.scm.azurewebsites.net/webssh/host
```

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

```
root@9e933156516f:~# service --status-all
[+] apache2
[-] bootlogs
[-] bootmisc.sh
[-] checkfs.sh
[-] checkroot-bootclean.sh
[-] checkroot.sh
[-] hostname.sh
[?] hwclock.sh
[-] killprocs
[-] motd
[-] mountall-bootclean.sh
[-] mountall.sh
[-] mountdevsubfs.sh
[-] mountkernfs.sh
[-] mountnfs-bootclean.sh
[-] mountnfs.sh
[-] mysql
[-] procps
[-] rc.local
[-] rmmlogin
[-] sendsigs
[+] ssh
[+] udev
[?] udev-finish
[-] umountfs
[-] umountnfs.sh
[-] umountroot
[-] urandom
root@9e933156516f:~#
```

ssh://root@[REDACTED]:2222 SSH CONNECTION ESTABLISHED

#### NOTE

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

When you're successfully connected to the SSH session, you should see the message "SSH CONNECTION ESTABLISHED" at the bottom of the window. If you see errors such as "SSH\_CONNECTION\_CLOSED" or a message that the container is restarting, an error may be preventing the app container from starting. See [Troubleshooting](#) for steps to investigate possible issues.

## Troubleshooting

In general, the first step in troubleshooting is to use App Service Diagnostics:

1. On the Azure portal for your web app, select **Diagnose and solve problems** from the left menu.
2. Select **Availability and performance**.
3. Examine the information in the **Application Logs**, **Container crash**, and **Container Issues** options, where the most common issues will appear.

Next, examine both the [deployment logs](#) and the [app logs](#) for any error messages. These logs often identify specific issues that can prevent app deployment or app startup. For example, the build can fail if your

*requirements.txt* file has the wrong filename or isn't present in your project root folder.

The following sections provide additional guidance for specific issues.

- [App doesn't appear - default app shows](#)
- [App doesn't appear - "service unavailable" message](#)
- [Could not find setup.py or requirements.txt](#)
- [ModuleNotFoundError on startup](#)
- [Database is locked](#)
- [Passwords don't appear in SSH session when typed](#)
- [Commands in the SSH session appear to be cut off](#)
- [Static assets don't appear in a Django app](#)
- [Fatal SSL Connection is Required](#)

#### **App doesn't appear**

- **You see the default app after deploying your own app code.** The [default app](#) appears because you either haven't deployed your app code to App Service, or App Service failed to find your app code and ran the default app instead.
  - Restart the App Service, wait 15-20 seconds, and check the app again.
  - Be sure you're using App Service for Linux rather than a Windows-based instance. From the Azure CLI, run the command

```
az webapp show --resource-group <resource-group-name> --name <app-name> --query kind
```

, replacing `<resource-group-name>` and `<app-name>` accordingly. You should see `app,linux` as output; otherwise, recreate the App Service and choose Linux.
  - Use [SSH](#) to connect directly to the App Service container and verify that your files exist under `site/wwwroot`. If your files don't exist, use the following steps:
    1. Create an app setting named `SCM_DO_BUILD_DURING_DEPLOYMENT` with the value of 1, redeploy your code, wait a few minutes, then try to access the app again. For more information on creating app settings, see [Configure an App Service app in the Azure portal](#).
    2. Review your deployment process, [check the deployment logs](#), correct any errors, and redeploy the app.
  - If your files exist, then App Service wasn't able to identify your specific startup file. Check that your app is structured as App Service expects for [Django](#) or [Flask](#), or use a [custom startup command](#).
- **You see the message "Service Unavailable" in the browser.** The browser has timed out waiting for a response from App Service, which indicates that App Service started the Gunicorn server, but the app itself did not start. This condition could indicate that the Gunicorn arguments are incorrect, or that there's an error in the app code.
  - Refresh the browser, especially if you're using the lowest pricing tiers in your App Service Plan. The app may take longer to start up when using free tiers, for example, and becomes responsive after you refresh the browser.
  - Check that your app is structured as App Service expects for [Django](#) or [Flask](#), or use a [custom startup command](#).
  - Examine the [app log stream](#) for any error messages. The logs will show any errors in the app code.

#### **Could not find setup.py or requirements.txt**

- **The log stream shows "Could not find setup.py or requirements.txt; Not running pip install."**: The Oryx build process failed to find your *requirements.txt* file.

- Connect to the web app's container via [SSH](#) and verify that `requirements.txt` is named correctly and exists directly under `site/wwwroot`. If it doesn't exist, make sure the file exists in your repository and is included in your deployment. If it exists in a separate folder, move it to the root.

#### ModuleNotFoundError when app starts

If you see an error like `ModuleNotFoundError: No module named 'example'`, this means that Python could not find one or more of your modules when the application started. This most often occurs if you deploy your virtual environment with your code. Virtual environments are not portable, so a virtual environment should not be deployed with your application code. Instead, let Oryx create a virtual environment and install your packages on the web app by creating an app setting, `SCM_DO_BUILD_DURING_DEPLOYMENT`, and setting it to `1`. This will force Oryx to install your packages whenever you deploy to App Service. For more information, please see [this article on virtual environment portability](#).

#### Database is locked

When attempting to run database migrations with a Django app, you may see "sqlite3.OperationalError: database is locked." The error indicates that your application is using a SQLite database for which Django is configured by default, rather than using a cloud database such as PostgreSQL for Azure.

Check the `DATABASES` variable in the app's `settings.py` file to ensure that your app is using a cloud database instead of SQLite.

If you're encountering this error with the sample in [Tutorial: Deploy a Django web app with PostgreSQL](#), check that you completed the steps in [Configure environment variables to connect the database](#).

#### Other issues

- **Passwords don't appear in the SSH session when typed:** For security reasons, the SSH session keeps your password hidden as you type. The characters are being recorded, however, so type your password as usual and press `Enter` when done.
- **Commands in the SSH session appear to be cut off:** The editor may not be word-wrapping commands, but they should still run correctly.
- **Static assets don't appear in a Django app:** Ensure that you have enabled the [whitenoise module](#)
- **You see the message, "Fatal SSL Connection is Required":** Check any usernames and passwords used to access resources (such as databases) from within the app.

## More resources:

- [Tutorial: Python app with PostgreSQL](#)
- [Tutorial: Deploy from private container repository](#)
- [App Service Linux FAQ](#)
- [Environment variables and app settings reference](#)

# Configure a Java app for Azure App Service

11/2/2021 • 34 minutes to read • [Edit Online](#)

Azure App Service lets Java developers quickly build, deploy, and scale their Java SE, Tomcat, and JBoss EAP web applications on a fully managed service. Deploy applications with Maven plugins, from the command line, or in editors like IntelliJ, Eclipse, or Visual Studio Code.

This guide provides key concepts and instructions for Java developers using App Service. If you've never used Azure App Service, you should read through the [Java quickstart](#) first. General questions about using App Service that aren't specific to Java development are answered in the [App Service FAQ](#).

## Show Java version

To show the current Java version, run the following command in the [Cloud Shell](#):

```
az webapp config show --name <app-name> --resource-group <resource-group-name> --query "[javaVersion, javaContainer, javaContainerVersion]"
```

To show all supported Java versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes | grep java
```

To show the current Java version, run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query linuxFxVersion
```

To show all supported Java versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes --linux | grep "JAVA\|TOMCAT\|JBOSSEAP"
```

## Deploying your app

### Build Tools

#### Maven

With the [Maven Plugin for Azure Web Apps](#), you can prepare your Maven Java project for Azure Web App easily with one command in your project root:

```
mvn com.microsoft.azure:azure-webapp-maven-plugin:2.2.0:config
```

This command adds a `azure-webapp-maven-plugin` plugin and related configuration by prompting you to select an existing Azure Web App or create a new one. Then you can deploy your Java app to Azure using the following command:

```
mvn package azure-webapp:deploy
```

Here is a sample configuration in `pom.xml`:

```

<plugin>
 <groupId>com.microsoft.azure</groupId>
 <artifactId>azure-webapp-maven-plugin</artifactId>
 <version>2.2.0</version>
 <configuration>
 <subscriptionId>111111-11111-111111111111</subscriptionId>
 <resourceGroup>spring-boot-xxxxxxxxxx-rg</resourceGroup>
 <appName>spring-boot-xxxxxxxxxx</appName>
 <pricingTier>B2</pricingTier>
 <region>westus</region>
 <runtime>
 <os>Linux</os>
 <webContainer>Java SE</webContainer>
 <javaVersion>Java 11</javaVersion>
 </runtime>
 <deployment>
 <resources>
 <resource>
 <type>jar</type>
 <directory>${project.basedir}/target</directory>
 <includes>
 <include>*.jar</include>
 </includes>
 </resource>
 </resources>
 </deployment>
 </configuration>
</plugin>

```

## Gradle

1. Setup the [Gradle Plugin for Azure Web Apps](#) by adding the plugin to your `build.gradle` :

```

plugins {
 id "com.microsoft.azure.azurewebapp" version "1.2.0"
}

```

2. Configure your Web App details, corresponding Azure resources will be created if not exist. Here is a sample configuration, for details, refer to this [document](#).

```

azurewebapp {
 subscription = '<your subscription id>'
 resourceGroup = '<your resource group>'
 appName = '<your app name>'
 pricingTier = '<price tier like 'P1v2'>'
 region = '<region like 'westus'>'
 runtime {
 os = 'Linux'
 webContainer = 'Tomcat 9.0' // or 'Java SE' if you want to run an executable jar
 javaVersion = 'Java 8'
 }
 appSettings {
 <key> = <value>
 }
 auth {
 type = 'azure_cli' // support azure_cli, oauth2, device_code and service_principal
 }
}

```

3. Deploy with one command.

```
gradle azureWebAppDeploy
```

## IDEs

Azure provides seamless Java App Service development experience in popular Java IDEs, including:

- *VS Code*: [Java Web Apps with Visual Studio Code](#)
- *IntelliJ IDEA*: [Create a Hello World web app for Azure App Service using IntelliJ](#)
- *Eclipse*: [Create a Hello World web app for Azure App Service using Eclipse](#)

## Kudu API

### Java SE

To deploy jar files to Java SE, use the `/api/publish/` endpoint of the Kudu site. For more information on this API, see [this documentation](#).

#### NOTE

Your jar application must be named `app.jar` for App Service to identify and run your application. The Maven Plugin (mentioned above) will automatically rename your application for you during deployment. If you do not wish to rename your JAR to `app.jar`, you can upload a shell script with the command to run your jar app. Paste the absolute path to this script in the [Startup File](#) textbox in the Configuration section of the portal. The startup script does not run from the directory into which it is placed. Therefore, always use absolute paths to reference files in your startup script (for example: `java -jar /home/myapp/myapp.jar`).

### Tomcat

To deploy .war files to Tomcat, use the `/api/wardeploy/` endpoint to POST your archive file. For more information on this API, see [this documentation](#).

### JBoss EAP

To deploy .war files to JBoss, use the `/api/wardeploy/` endpoint to POST your archive file. For more information on this API, see [this documentation](#).

To deploy .ear files, [use FTP](#). Your .ear application will be deployed to the context root defined in your application's configuration. For example, if the context root of your app is `<context-root>myapp</context-root>`, then you can browse the site at the `/myapp` path: `http://my-app-name.azurewebsites.net/myapp`. If you want your web app to be served in the root path, ensure that your app sets the context root to the root path: `<context-root>/</context-root>`. For more information, see [Setting the context root of a web application](#).

Do not deploy your .war or jar using FTP. The FTP tool is designed to upload startup scripts, dependencies, or other runtime files. It is not the optimal choice for deploying web apps.

## Logging and debugging apps

Performance reports, traffic visualizations, and health checkups are available for each app through the Azure portal. For more information, see [Azure App Service diagnostics overview](#).

### Stream diagnostic logs

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the

previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

#### NOTE

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

```
az webapp log config --name <app-name> --resource-group <resource-group-name> --docker-container-logging filesystem
```

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

Once container logging is turned on, run the following command to see the log stream:

```
az webapp log tail --name <app-name> --resource-group <resource-group-name>
```

If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type `Ctrl+C`.

You can also inspect the log files in a browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

For more information, see [Stream logs in Cloud Shell](#).

#### SSH console access

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

```
https://<app-name>.scm.azurewebsites.net/webssh/host
```

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

The screenshot shows an SSH session in a browser window titled "SSH". The command "service --status-all" is run, displaying a list of services and their status. The output includes services like apache2, bootlogs, bootmisc.sh, checkfs.sh, checkroot-bootclean.sh, checkroot.sh, hostname.sh, hwclock.sh, killprocs, motd, mountall-bootclean.sh, mountall.sh, mountdevsubfs.sh, mountkernfs.sh, mountnfs-bootclean.sh, mountnfs.sh, mysql, procps, rc.local, rmmlogin, sendsigs, ssh, udev, udev-finish, umountfs, umountnfs.sh, umountroot, and urandom. The session ends with "root@9e933156516f:~#". Below the terminal window, a status bar shows "ssh://root@<ip>:2222" and "SSH CONNECTION ESTABLISHED".

```
root@9e933156516f:~# service --status-all
[+] apache2
[-] bootlogs
[-] bootmisc.sh
[-] checkfs.sh
[-] checkroot-bootclean.sh
[-] checkroot.sh
[-] hostname.sh
[?] hwclock.sh
[-] killprocs
[-] motd
[-] mountall-bootclean.sh
[-] mountall.sh
[-] mountdevsubfs.sh
[-] mountkernfs.sh
[-] mountnfs-bootclean.sh
[-] mountnfs.sh
[-] mysql
[-] procps
[-] rc.local
[-] rmmlogin
[-] sendsigs
[+] ssh
[+] udev
[?] udev-finish
[-] umountfs
[-] umountnfs.sh
[-] umountroot
[-] urandom
root@9e933156516f:~#
```

ssh://root@<ip>:2222 SSH CONNECTION ESTABLISHED

#### NOTE

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

#### Troubleshooting tools

The built-in Java images are based on the [Alpine Linux](#) operating system. Use the `apk` package manager to install any troubleshooting tools or commands.

#### Flight Recorder

All Java runtimes on App Service using the Azul JVMs come with the Zulu Flight Recorder. You can use this to record JVM, system, and application events and troubleshoot problems in your Java applications.

#### Timed Recording

To take a timed recording, you'll need the PID (Process ID) of the Java application. To find the PID, open a browser to your web app's SCM site at <https://<your-site-name>.scm.azurewebsites.net/ProcessExplorer/>. This page shows the running processes in your web app. Find the process named "java" in the table and copy the corresponding PID (Process ID).

Next, open the **Debug Console** in the top toolbar of the SCM site and run the following command. Replace `<pid>` with the process ID you copied earlier. This command will start a 30-second profiler recording of your Java application and generate a file named `timed_recording_example.jfr` in the `D:\home` directory.

```
jcmd <pid> JFR.start name=TimedRecording settings=profile duration=30s
filename="D:\home\timed_recording_example.JFR"
```

SSH into your App Service and run the `jcmd` command to see a list of all the Java processes running. In

addition to jcmb itself, you should see your Java application running with a process ID number (pid).

```
078990bbcd11:/home# jcmb
Picked up JAVA_TOOL_OPTIONS: -Djava.net.preferIPv4Stack=true
147 sun.tools.jcmb.JCmb
116 /home/site/wwwroot/app.jar
```

Execute the command below to start a 30-second recording of the JVM. This will profile the JVM and create a JFR file named *jfr\_example.jfr* in the home directory. (Replace 116 with the pid of your Java app.)

```
jcmb 116 JFR.start name=MyRecording settings=profile duration=30s filename="/home/jfr_example.jfr"
```

During the 30-second interval, you can validate the recording is taking place by running `jcmb 116 JFR.check`. This will show all recordings for the given Java process.

#### Continuous Recording

You can use Zulu Flight Recorder to continuously profile your Java application with minimal impact on runtime performance. To do so, run the following Azure CLI command to create an App Setting named JAVA\_OPTS with the necessary configuration. The contents of the JAVA\_OPTS App Setting are passed to the `java` command when your app is started.

```
az webapp config appsettings set -g <your_resource_group> -n <your_app_name> --settings JAVA_OPTS=-XX:StartFlightRecording=disk=true,name=continuous_recording,dumponexit=true,maxsize=1024m,maxage=1d
```

Once the recording has started, you can dump the current recording data at any time using the `JFR.dump` command.

```
jcmb <pid> JFR.dump name=continuous_recording filename="/home/recording1.jfr"
```

#### Analyze .jfr files

Use [FTPS](#) to download your JFR file to your local machine. To analyze the JFR file, download and install [Zulu Mission Control](#). For instructions on Zulu Mission Control, see the [Azul documentation](#) and the [installation instructions](#).

#### App logging

Enable [application logging](#) through the Azure portal or [Azure CLI](#) to configure App Service to write your application's standard console output and standard console error streams to the local filesystem or Azure Blob Storage. Logging to the local App Service filesystem instance is disabled 12 hours after it is configured. If you need longer retention, configure the application to write output to a Blob storage container. Your Java and Tomcat app logs can be found in the `/home/LogFiles/Application/` directory.

Enable [application logging](#) through the Azure portal or [Azure CLI](#) to configure App Service to write your application's standard console output and standard console error streams to the local filesystem or Azure Blob Storage. If you need longer retention, configure the application to write output to a Blob storage container. Your Java and Tomcat app logs can be found in the `/home/LogFiles/Application/` directory.

Azure Blob Storage logging for Linux based App Services can only be configured using [Azure Monitor](#)

If your application uses [Logback](#) or [Log4j](#) for tracing, you can forward these traces for review into Azure Application Insights using the logging framework configuration instructions in [Explore Java trace logs in Application Insights](#).

## Customization and tuning

Azure App Service for Linux supports out of the box tuning and customization through the Azure portal and CLI. Review the following articles for non-Java-specific web app configuration:

- [Configure app settings](#)
- [Set up a custom domain](#)
- [Configure TLS/SSL bindings](#)
- [Add a CDN](#)
- [Configure the Kudu site](#)

## Set Java runtime options

To set allocated memory or other JVM runtime options, create an [app setting](#) named `JAVA_OPTS` with the options. App Service passes this setting as an environment variable to the Java runtime when it starts.

In the Azure portal, under **Application Settings** for the web app, create a new app setting named `JAVA_OPTS` for Java SE or `CATALINA_OPTS` for Tomcat that includes other settings, such as `-Xms512m -Xmx1204m`.

To configure the app setting from the Maven plugin, add setting/value tags in the Azure plugin section. The following example sets a specific minimum and maximum Java heap size:

```
<appSettings>
 <property>
 <name>JAVA_OPTS</name>
 <value>-Xms512m -Xmx1204m</value>
 </property>
</appSettings>
```

Developers running a single application with one deployment slot in their App Service plan can use the following options:

- B1 and S1 instances: `-Xms1024m -Xmx1024m`
- B2 and S2 instances: `-Xms3072m -Xmx3072m`
- B3 and S3 instances: `-Xms6144m -Xmx6144m`
- P1v2 instances: `-Xms3072m -Xmx3072m`
- P2v2 instances: `-Xms6144m -Xmx6144m`
- P3v2 instances: `-Xms12800m -Xmx12800m`
- P1v3 instances: `-Xms6656m -Xmx6656m`
- P2v3 instances: `-Xms14848m -Xmx14848m`
- P3v3 instances: `-Xms30720m -Xmx30720m`
- I1 instances: `-Xms3072m -Xmx3072m`
- I2 instances: `-Xms6144m -Xmx6144m`
- I3 instances: `-Xms12800m -Xmx12800m`
- I1v2 instances: `-Xms6656m -Xmx6656m`
- I2v2 instances: `-Xms14848m -Xmx14848m`
- I3v2 instances: `-Xms30720m -Xmx30720m`

When tuning application heap settings, review your App Service plan details and take into account multiple applications and deployment slot needs to find the optimal allocation of memory.

## Turn on web sockets

Turn on support for web sockets in the Azure portal in the **Application settings** for the application. You'll need to restart the application for the setting to take effect.

Turn on web socket support using the Azure CLI with the following command:

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --web-sockets-enabled true
```

Then restart your application:

```
az webapp stop --name <app-name> --resource-group <resource-group-name>
az webapp start --name <app-name> --resource-group <resource-group-name>
```

## Set default character encoding

In the Azure portal, under **Application Settings** for the web app, create a new app setting named `JAVA_OPTS` with value `-Dfile.encoding=UTF-8`.

Alternatively, you can configure the app setting using the App Service Maven plugin. Add the setting name and value tags in the plugin configuration:

```
<appSettings>
 <property>
 <name>JAVA_OPTS</name>
 <value>-Dfile.encoding=UTF-8</value>
 </property>
</appSettings>
```

## Pre-Compile JSP files

To improve performance of Tomcat applications, you can compile your JSP files before deploying to App Service. You can use the [Maven plugin](#) provided by Apache Sling, or using this [Ant build file](#).

# Secure applications

Java applications running in App Service have the same set of [security best practices](#) as other applications.

## Authenticate users (Easy Auth)

Set up app authentication in the Azure portal with the **Authentication and Authorization** option. From there, you can enable authentication using Azure Active Directory or social logins like Facebook, Google, or GitHub. Azure portal configuration only works when configuring a single authentication provider. For more information, see [Configure your App Service app to use Azure Active Directory login](#) and the related articles for other identity providers. If you need to enable multiple sign-in providers, follow the instructions in the [customize sign-ins and sign-outs](#) article.

### Java SE

Spring Boot developers can use the [Azure Active Directory Spring Boot starter](#) to secure applications using familiar Spring Security annotations and APIs. Be sure to increase the maximum header size in your `application.properties` file. We suggest a value of `16384`.

### Tomcat

Your Tomcat application can access the user's claims directly from the servlet by casting the `Principal` object to a `Map` object. The `Map` object will map each claim type to a collection of the claims for that type. In the code below, `request` is an instance of `HttpServletRequest`.

```
Map<String, Collection<String>> map = (Map<String, Collection<String>>) request.getUserPrincipal();
```

Now you can inspect the `Map` object for any specific claim. For example, the following code snippet iterates through all the claim types and prints the contents of each collection.

```

for (Object key : map.keySet()) {
 Object value = map.get(key);
 if (value != null && value instanceof Collection) {
 Collection claims = (Collection) value;
 for (Object claim : claims) {
 System.out.println(claims);
 }
 }
}

```

To sign out users, use the `/.auth/ext/logout` path. To perform other actions, see the documentation on [Customize sign-ins and sign-outs](#). There is also official documentation on the Tomcat [HttpServletRequest interface](#) and its methods. The following servlet methods are also hydrated based on your App Service configuration:

```

public boolean isSecure()
public String getRemoteAddr()
public String getRemoteHost()
public String getScheme()
public int getServerPort()

```

To disable this feature, create an Application Setting named `WEBSITE_AUTH_SKIP_PRINCIPAL` with a value of `1`. To disable all servlet filters added by App Service, create a setting named `WEBSITE_SKIP_FILTERS` with a value of `1`.

## Configure TLS/SSL

Follow the instructions in the [Secure a custom DNS name with an TLS/SSL binding in Azure App Service](#) to upload an existing TLS/SSL certificate and bind it to your application's domain name. By default your application will still allow HTTP connections-follow the specific steps in the tutorial to enforce TLS/SSL.

## Use KeyVault References

[Azure KeyVault](#) provides centralized secret management with access policies and audit history. You can store secrets (such as passwords or connection strings) in KeyVault and access these secrets in your application through environment variables.

First, follow the instructions for [granting your app access to Key Vault](#) and [making a KeyVault reference to your secret in an Application Setting](#). You can validate that the reference resolves to the secret by printing the environment variable while remotely accessing the App Service terminal.

To inject these secrets in your Spring or Tomcat configuration file, use environment variable injection syntax ( `${MY_ENV_VAR}` ). For Spring configuration files, see this documentation on [externalized configurations](#).

## Use the Java Key Store

By default, any public or private certificates [uploaded to App Service Linux](#) will be loaded into the respective Java Key Stores as the container starts. After uploading your certificate, you will need to restart your App Service for it to be loaded into the Java Key Store. Public certificates are loaded into the Key Store at  `${JAVA_HOME}/jre/lib/security/cacerts` , and private certificates are stored in  `${JAVA_HOME}/lib/security/client.jks` .

More configuration may be necessary for encrypting your JDBC connection with certificates in the Java Key Store. Refer to the documentation for your chosen JDBC driver.

- [PostgreSQL](#)
- [SQL Server](#)
- [MySQL](#)
- [MongoDB](#)
- [Cassandra](#)

## Initialize the Java Key Store

To initialize the `import java.security.KeyStore` object, load the keystore file with the password. The default password for both key stores is `changeit`.

```
KeyStore keyStore = KeyStore.getInstance("jks");
keyStore.load(
 new FileInputStream(System.getenv("JAVA_HOME")+"/lib/security/cacerts"),
 "changeit".toCharArray());

KeyStore keyStore = KeyStore.getInstance("pkcs12");
keyStore.load(
 new FileInputStream(System.getenv("JAVA_HOME")+"/lib/security/client.jks"),
 "changeit".toCharArray());
```

## Manually load the key store

You can load certificates manually to the key store. Create an app setting, `SKIP_JAVA_KEYSTORE_LOAD`, with a value of `1` to disable App Service from loading the certificates into the key store automatically. All public certificates uploaded to App Service via the Azure portal are stored under `/var/ssl/certs/`. Private certificates are stored under `/var/ssl/private/`.

You can interact or debug the Java Key Tool by [opening an SSH connection](#) to your App Service and running the command `keytool`. See the [Key Tool documentation](#) for a list of commands. For more information on the KeyStore API, refer to [the official documentation](#).

## Configure APM platforms

This section shows how to connect Java applications deployed on Azure App Service with Azure Monitor application insights, NewRelic, and AppDynamics application performance monitoring (APM) platforms.

### Configure Application Insights

Azure Monitor application insights is a cloud native application monitoring service that enables customers to observe failures, bottlenecks, and usage patterns to improve application performance and reduce mean time to resolution (MTTR). With a few clicks or CLI commands, you can enable monitoring for your Node.js or Java apps, auto-collecting logs, metrics, and distributed traces, eliminating the need for including an SDK in your app.

#### Azure portal

To enable Application Insights from the Azure portal, go to **Application Insights** on the left-side menu and select **Turn on Application Insights**. By default, a new application insights resource of the same name as your Web App will be used. You can choose to use an existing application insights resource, or change the name. Click **Apply** at the bottom

#### Azure CLI

To enable via the Azure CLI, you will need to create an Application Insights resource and set a couple app settings on the Azure portal to connect Application Insights to your web app.

1. Enable the Applications Insights extension

```
az extension add -n application-insights
```

2. Create an Application Insights resource using the CLI command below. Replace the placeholders with your desired resource name and group.

```
az monitor app-insights component create --app <resource-name> -g <resource-group> --location westus2
--kind web --application-type web
```

Note the values for `connectionString` and `instrumentationKey`, you will need these values in the next step.

To retrieve a list of other locations, run `az account list-locations`.

- Set the instrumentation key, connection string, and monitoring agent version as app settings on the web app. Replace `<instrumentationKey>` and `<connectionString>` with the values from the previous step.

```
az webapp config appsettings set -n <webapp-name> -g <resource-group> --settings
"APPINSIGHTS_INSTRUMENTATIONKEY=<instrumentationKey>" "APPLICATIONINSIGHTS_CONNECTION_STRING=
<connectionString>" "ApplicationInsightsAgent_EXTENSION_VERSION=~3"
"XDT_MicrosoftApplicationInsights_Mode=default" "XDT_MicrosoftApplicationInsights_Java=1"
```

- Set the instrumentation key, connection string, and monitoring agent version as app settings on the web app. Replace `<instrumentationKey>` and `<connectionString>` with the values from the previous step.

```
az webapp config appsettings set -n <webapp-name> -g <resource-group> --settings
"APPINSIGHTS_INSTRUMENTATIONKEY=<instrumentationKey>" "APPLICATIONINSIGHTS_CONNECTION_STRING=
<connectionString>" "ApplicationInsightsAgent_EXTENSION_VERSION=~3"
"XDT_MicrosoftApplicationInsights_Mode=default"
```

## Configure New Relic

- Create a NewRelic account at [NewRelic.com](#)
- Download the Java agent from NewRelic, it will have a file name similar to `newrelic-java-x.x.x.zip`.
- Copy your license key, you'll need it to configure the agent later.
- [SSH into your App Service instance](#) and create a new directory `/home/site/wwwroot/apm`.
- Upload the unpacked NewRelic Java agent files into a directory under `/home/site/wwwroot/apm`. The files for your agent should be in `/home/site/wwwroot/apm/newrelic`.
- Modify the YAML file at `/home/site/wwwroot/apm/newrelic/newrelic.yml` and replace the placeholder license value with your own license key.
- In the Azure portal, browse to your application in App Service and create a new Application Setting.
  - For Java SE apps, create an environment variable named `JAVA_OPTS` with the value  
`-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.
  - For Tomcat, create an environment variable named `CATALINA_OPTS` with the value  
`-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.
- Create a NewRelic account at [NewRelic.com](#)
- Download the Java agent from NewRelic, it will have a file name similar to `newrelic-java-x.x.x.zip`.
- Copy your license key, you'll need it to configure the agent later.
- [SSH into your App Service instance](#) and create a new directory `/home/site/wwwroot/apm`.
- Upload the unpacked NewRelic Java agent files into a directory under `/home/site/wwwroot/apm`. The files for your agent should be in `/home/site/wwwroot/apm/newrelic`.
- Modify the YAML file at `/home/site/wwwroot/apm/newrelic/newrelic.yml` and replace the placeholder license value with your own license key.
- In the Azure portal, browse to your application in App Service and create a new Application Setting.

- For **Java SE** apps, create an environment variable named **JAVA\_OPTS** with the value  
`-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.
- For **Tomcat**, create an environment variable named **CATALINA\_OPTS** with the value  
`-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.

If you already have an environment variable for **JAVA\_OPTS** or **CATALINA\_OPTS**, append the `-javaagent:/...` option to the end of the current value.

## Configure AppDynamics

1. Create an AppDynamics account at [AppDynamics.com](#)
2. Download the Java agent from the AppDynamics website, the file name will be similar to *AppServerAgent-x.x.x.xxxxx.zip*
3. Use the [Kudu console](#) to create a new directory */home/site/wwwroot/apm*.
4. Upload the Java agent files into a directory under */home/site/wwwroot/apm*. The files for your agent should be in */home/site/wwwroot/apm/appdynamics*.
5. In the Azure portal, browse to your application in App Service and create a new Application Setting.

- For **Java SE** apps, create an environment variable named **JAVA\_OPTS** with the value  
`-javaagent:/home/site/wwwroot/apm/appdynamics/javaagent.jar -Dappdynamics.agent.applicationName=<app-name>`  
 where **<app-name>** is your App Service name.
- For **Tomcat** apps, create an environment variable named **CATALINA\_OPTS** with the value  
`-javaagent:/home/site/wwwroot/apm/appdynamics/javaagent.jar -Dappdynamics.agent.applicationName=<app-name>`  
 where **<app-name>** is your App Service name.

1. Create an AppDynamics account at [AppDynamics.com](#)
2. Download the Java agent from the AppDynamics website, the file name will be similar to *AppServerAgent-x.x.x.xxxxx.zip*
3. [SSH into your App Service instance](#) and create a new directory */home/site/wwwroot/apm*.
4. Upload the Java agent files into a directory under */home/site/wwwroot/apm*. The files for your agent should be in */home/site/wwwroot/apm/appdynamics*.
5. In the Azure portal, browse to your application in App Service and create a new Application Setting.

- For **Java SE** apps, create an environment variable named **JAVA\_OPTS** with the value  
`-javaagent:/home/site/wwwroot/apm/appdynamics/javaagent.jar -Dappdynamics.agent.applicationName=<app-name>`  
 where **<app-name>** is your App Service name.
- For **Tomcat** apps, create an environment variable named **CATALINA\_OPTS** with the value  
`-javaagent:/home/site/wwwroot/apm/appdynamics/javaagent.jar -Dappdynamics.agent.applicationName=<app-name>`  
 where **<app-name>** is your App Service name.

### NOTE

If you already have an environment variable for **JAVA\_OPTS** or **CATALINA\_OPTS**, append the `-javaagent:/...` option to the end of the current value.

# Configure data sources

## Java SE

To connect to data sources in Spring Boot applications, we suggest creating connection strings and injecting them into your *application.properties* file.

1. In the "Configuration" section of the App Service page, set a name for the string, paste your JDBC connection string into the value field, and set the type to "Custom". You can optionally set this connection string as slot setting.

This connection string is accessible to our application as an environment variable named `CUSTOMCONNSTR_<your-string-name>`. For example, the connection string we created above will be named `CUSTOMCONNSTR_exampledbs`.

2. In your *application.properties* file, reference this connection string with the environment variable name. For our example, we would use the following.

```
app.datasource.url=${CUSTOMCONNSTR_exampledbs}
```

See the [Spring Boot documentation on data access](#) and [externalized configurations](#) for more information on this topic.

## Tomcat

These instructions apply to all database connections. You will need to fill placeholders with your chosen database's driver class name and JAR file. Provided is a table with class names and driver downloads for common databases.

DATABASE	DRIVER CLASS NAME	JDBC DRIVER
PostgreSQL	<code>org.postgresql.Driver</code>	<a href="#">Download</a>
MySQL	<code>com.mysql.jdbc.Driver</code>	<a href="#">Download</a> (Select "Platform Independent")
SQL Server	<code>com.microsoft.sqlserver.jdbc.SQLServerDriver</code>	<a href="#">Download</a>

To configure Tomcat to use Java Database Connectivity (JDBC) or the Java Persistence API (JPA), first customize the `CATALINA_OPTS` environment variable that is read in by Tomcat at start-up. Set these values through an app setting in the [App Service Maven plugin](#):

```
<appSettings>
<property>
 <name>CATALINA_OPTS</name>
 <value>"$CATALINA_OPTS -Ddbuser=${DBUSER} -Ddbpassword=${DBPASSWORD} -DconnURL=${CONNURL}"</value>
</property>
</appSettings>
```

Or set the environment variables in the [Configuration > Application Settings](#) page in the Azure portal.

Next, determine if the data source should be available to one application or to all applications running on the Tomcat servlet.

## Application-level data sources

1. Create a *context.xml* file in the *META-INF*/directory of your project. Create the *META-INF*/directory if it does not exist.

2. In `context.xml`, add a `<Context>` element to link the data source to a JNDI address. Replace the `driverClassName` placeholder with your driver's class name from the table above.

```
<Context>
 <Resource
 name="jdbc/dbconnection"
 type="javax.sql.DataSource"
 url="${dbuser}"
 driverClassName=<insert your driver class name>
 username="${dbpassword}"
 password="${connURL}"
 />
</Context>
```

3. Update your application's `web.xml` to use the data source in your application.

```
<resource-env-ref>
 <resource-env-ref-name>jdbc/dbconnection</resource-env-ref-name>
 <resource-env-ref-type>javax.sql.DataSource</resource-env-ref-type>
</resource-env-ref>
```

#### Shared server-level resources

Tomcat installations on App Service on Windows exist in shared space on the App Service Plan. You can't directly modify a Tomcat installation for server-wide configuration. To make server-level configuration changes to your Tomcat installation, you must copy Tomcat to a local folder, in which you can modify Tomcat's configuration.

##### Automate creating custom Tomcat on app start

You can use a startup script to perform actions before a web app starts. The startup script for customizing Tomcat needs to complete the following steps:

1. Check whether Tomcat was already copied and configured locally. If it was, the startup script can end here.
2. Copy Tomcat locally.
3. Make the required configuration changes.
4. Indicate that configuration was successfully completed.

For Windows sites, create a file named `startup.cmd` or `startup.ps1` in the `wwwroot` directory. This will automatically be executed before the Tomcat server starts.

Here's a PowerShell script that completes these steps:

```

Check for marker file indicating that config has already been done
if(Test-Path "$Env:LOCAL_EXPANDED\tomcat\config_done_marker"){
 return 0
}

Delete previous Tomcat directory if it exists
In case previous config could not be completed or a new config should be forcefully installed
if(Test-Path "$Env:LOCAL_EXPANDED\tomcat"){
 Remove-Item "$Env:LOCAL_EXPANDED\tomcat" --recurse
}

Copy Tomcat to local
Using the environment variable $AZURE_TOMCAT90_HOME uses the 'default' version of Tomcat
Copy-Item -Path "$Env:AZURE_TOMCAT90_HOME*" -Destination "$Env:LOCAL_EXPANDED\tomcat" -Recurse

Perform the required customization of Tomcat
{... customization ...}

Mark that the operation was a success
New-Item -Path "$Env:LOCAL_EXPANDED\tomcat\config_done_marker" -ItemType File

```

#### Transforms

A common use case for customizing a Tomcat version is to modify the `server.xml`, `context.xml`, or `web.xml` Tomcat configuration files. App Service already modifies these files to provide platform features. To continue to use these features, it's important to preserve the content of these files when you make changes to them. To accomplish this, we recommend that you use an [XSL transformation \(XSLT\)](#). Use an XSL transform to make changes to the XML files while preserving the original contents of the file.

#### Example XSLT file

This example transform adds a new connector node to `server.xml`. Note the *Identity Transform*, which preserves the original contents of the file.

```

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" indent="yes"/>

<!-- Identity transform: this ensures that the original contents of the file are included in the new
file -->
<!-- Ensure that your transform files include this block -->
<xsl:template match="@* | node()" name="Copy">
 <xsl:copy>
 <xsl:apply-templates select="@* | node()"/>
 </xsl:copy>
</xsl:template>

<xsl:template match="@* | node()" mode="insertConnector">
 <xsl:call-template name="Copy" />
</xsl:template>

<xsl:template match="comment()[not(..//Connector[@scheme = 'https']) and
contains(., '&lt;Connector') and
contains(., 'scheme="https"') or
contains(., "scheme='https'")]">
 <xsl:value-of select=". disable-output-escaping="yes" />
</xsl:template>

<xsl:template match="Service[not(Connector[@scheme = 'https'] or
comment()[contains(., '&lt;Connector') and
contains(., 'scheme="https"') or
contains(., "scheme='https'")]]
)]
">
 <xsl:copy>
 <xsl:apply-templates select="@* | node()" mode="insertConnector" />
 </xsl:copy>
</xsl:template>

<!-- Add the new connector after the last existing Connector if there is one -->
<xsl:template match="Connector[last()]" mode="insertConnector">
 <xsl:call-template name="Copy" />

 <xsl:call-template name="AddConnector" />
</xsl:template>

<!-- ... or before the first Engine if there is no existing Connector -->
<xsl:template match="Engine[1][not(preceding-sibling::Connector)]"
mode="insertConnector">
 <xsl:call-template name="AddConnector" />

 <xsl:call-template name="Copy" />
</xsl:template>

<xsl:template name="AddConnector">
 <!-- Add new line -->
 <xsl:text>&#xa;</xsl:text>
 <!-- This is the new connector -->
 <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
 maxThreads="150" scheme="https" secure="true"
 keystoreFile="${user.home}/.keystore" keystorePass="changeit"
 clientAuth="false" sslProtocol="TLS" />
</xsl:template>

</xsl:stylesheet>
</pre>

```

Function for XSL transform

PowerShell has built-in tools for transforming XML files by using XSL transforms. The following script is an example function that you can use in `startup.ps1` to perform the transform:

```

function TransformXML{
 param ($xml, $xsl, $output)

 if (-not $xml -or -not $xsl -or -not $output)
 {
 return 0
 }

 Try
 {
 $xslt_settings = New-Object System.Xml.Xsl.XsltSettings;
 $XmlUrlResolver = New-Object System.Xml.XmlUrlResolver;
 $xslt_settings.EnableScript = 1;

 $xslt = New-Object System.Xml.Xsl.XslCompiledTransform;
 $xslt.Load($xsl,$xslt_settings,$XmlUrlResolver);
 $xslt.Transform($xml, $output);

 }

 Catch
 {
 $ErrorMessage = $_.Exception.Message
 $FailedItem = $_.Exception.ItemName
 Write-Host 'Error'$ErrorMessage':'$FailedItem':' $_.Exception;
 return 0
 }
 return 1
}

```

#### App settings

The platform also needs to know where your custom version of Tomcat is installed. You can set the installation's location in the `CATALINA_BASE` app setting.

You can use the Azure CLI to change this setting:

```

az webapp config appsettings set -g $MyResourceGroup -n $MyUniqueApp --settings
CATALINA_BASE="%LOCAL_EXPANDED%\tomcat"

```

Or, you can manually change the setting in the Azure portal:

1. Go to **Settings > Configuration > Application settings**.
2. Select **New Application Setting**.
3. Use these values to create the setting:
  - a. **Name:** `CATALINA_BASE`
  - b. **Value:** `"%LOCAL_EXPANDED%\tomcat"`

#### Example startup.ps1

The following example script copies a custom Tomcat to a local folder, performs an XSL transform, and indicates that the transform was successful:

```

Locations of xml and xsl files
$target_xml="$Env:LOCAL_EXPANDED\tomcat\conf\server.xml"
$target_xsl="$Env:HOME\site\server.xsl"

Define the transform function
Useful if transforming multiple files
function TransformXML{
 param ($xml, $xsl, $output)

 if (-not $xml -or -not $xsl -or -not $output)
 {
 return 0
 }

 Try
 {
 $xslt_settings = New-Object System.Xml.Xsl.XsltSettings;
 $XmlUrlResolver = New-Object System.Xml.XmlUrlResolver;
 $xslt_settings.EnableScript = 1;

 $xslt = New-Object System.Xml.Xsl.XslCompiledTransform;
 $xslt.Load($xsl,$xslt_settings,$XmlUrlResolver);
 $xslt.Transform($xml, $output);
 }

 Catch
 {
 $ErrorMessage = $_.Exception.Message
 $FailedItem = $_.Exception.ItemName
 echo 'Error'$ErrorMessage':'$FailedItem':' $_.Exception;
 return 0
 }
 return 1
}

$success = TransformXML -xml $target_xml -xsl $target_xsl -output $target_xml

Check for marker file indicating that config has already been done
if(Test-Path "$Env:LOCAL_EXPANDED\tomcat\config_done_marker"){
 return 0
}

Delete previous Tomcat directory if it exists
In case previous config could not be completed or a new config should be forcefully installed
if(Test-Path "$Env:LOCAL_EXPANDED\tomcat"){
 Remove-Item "$Env:LOCAL_EXPANDED\tomcat" --recurse
}

md -Path "$Env:LOCAL_EXPANDED\tomcat"

Copy Tomcat to local
Using the environment variable $AZURE_TOMCAT90_HOME uses the 'default' version of Tomcat
Copy-Item -Path "$Env:AZURE_TOMCAT90_HOME*" "$Env:LOCAL_EXPANDED\tomcat" -Recurse

Perform the required customization of Tomcat
$success = TransformXML -xml $target_xml -xsl $target_xsl -output $target_xml

Mark that the operation was a success if successful
if($success){
 New-Item -Path "$Env:LOCAL_EXPANDED\tomcat\config_done_marker" -ItemType File
}

```

#### Finalize configuration

Finally, we will place the driver JARs in the Tomcat classpath and restart your App Service. Ensure that the JDBC driver files are available to the Tomcat classloader by placing them in the `/home/tomcat/lib` directory. (Create this directory if it does not already exist.) To upload these files to your App Service instance, perform the

following steps:

1. In the [Cloud Shell](#), install the webapp extension:

```
az extension add --name webapp
```

2. Run the following CLI command to create an SSH tunnel from your local system to App Service:

```
az webapp remote-connection create --resource-group <resource-group-name> --name <app-name> --port <port-on-local-machine>
```

3. Connect to the local tunneling port with your SFTP client and upload the files to the `/home/tomcat/lib` folder.

Alternatively, you can use an FTP client to upload the JDBC driver. Follow these [instructions for getting your FTP credentials](#).

## Tomcat

These instructions apply to all database connections. You will need to fill placeholders with your chosen database's driver class name and JAR file. Provided is a table with class names and driver downloads for common databases.

DATABASE	DRIVER CLASS NAME	JDBC DRIVER
PostgreSQL	<code>org.postgresql.Driver</code>	<a href="#">Download</a>
MySQL	<code>com.mysql.jdbc.Driver</code>	<a href="#">Download</a> (Select "Platform Independent")
SQL Server	<code>com.microsoft.sqlserver.jdbc.SQLServerDriver</code>	<a href="#">Download</a>

To configure Tomcat to use Java Database Connectivity (JDBC) or the Java Persistence API (JPA), first customize the `CATALINA_OPTS` environment variable that is read in by Tomcat at start-up. Set these values through an app setting in the [App Service Maven plugin](#):

```
<appSettings>
 <property>
 <name>CATALINA_OPTS</name>
 <value>"$CATALINA_OPTS -Ddbuser=${DBUSER} -Ddbpassword=${DBPASSWORD} -DconnURL=${CONNURL}"</value>
 </property>
</appSettings>
```

Or set the environment variables in the [Configuration > Application Settings](#) page in the Azure portal.

Next, determine if the data source should be available to one application or to all applications running on the Tomcat servlet.

### Application-level data sources

1. Create a `context.xml` file in the `META-INF` directory of your project. Create the `META-INF` directory if it does not exist.
2. In `context.xml`, add a `context` element to link the data source to a JNDI address. Replace the `driverClassName` placeholder with your driver's class name from the table above.

```
<Context>
 <Resource
 name="jdbc/dbconnection"
 type="javax.sql.DataSource"
 url="${dbuser}"
 driverClassName=<insert your driver class name>
 username="${dbpassword}"
 password="${connURL}"
 />
</Context>
```

3. Update your application's *web.xml* to use the data source in your application.

```
<resource-env-ref>
 <resource-env-ref-name>jdbc/dbconnection</resource-env-ref-name>
 <resource-env-ref-type>javax.sql.DataSource</resource-env-ref-type>
</resource-env-ref>
```

#### Shared server-level resources

Adding a shared, server-level data source will require you to edit Tomcat's *server.xml*. First, upload a [startup script](#) and set the path to the script in **Configuration > Startup Command**. You can upload the startup script using [FTP](#).

Your startup script will make an [xsl transform](#) to the *server.xml* file and output the resulting xml file to `/usr/local/tomcat/conf/server.xml`. The startup script should install libxslt via apk. Your xsl file and startup script can be uploaded via FTP. Below is an example startup script.

```
Install libxslt. Also copy the transform file to /home/tomcat/conf/
apk add --update libxslt

Usage: xsltproc --output output.xml style.xsl input.xml
xsltproc --output /home/tomcat/conf/server.xml /home/tomcat/conf/transform.xsl
/usr/local/tomcat/conf/server.xml
```

An example xsl file is provided below. The example xsl file adds a new connector node to the Tomcat *server.xml*.

```

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="xml" indent="yes"/>

 <xsl:template match="@* | node()" name="Copy">
 <xsl:copy>
 <xsl:apply-templates select="@* | node()"/>
 </xsl:copy>
 </xsl:template>

 <xsl:template match="@* | node()" mode="insertConnector">
 <xsl:call-template name="Copy" />
 </xsl:template>

 <xsl:template match="comment()[not(..//Connector[@scheme = 'https']) and
 contains(., '<Connector') and
 (contains(., 'scheme="https"') or
 contains(., "schema='https'"))]">
 <xsl:value-of select="." disable-output-escaping="yes" />
 </xsl:template>

 <xsl:template match="Service[not(Connector[@scheme = 'https'] or
 comment()[contains(., '<Connector') and
 (contains(., 'scheme="https"') or
 contains(., "schema='https'"))]]]>
 <!-- Add the new connector after the last existing Connector if there is one -->
 <xsl:template match="Connector[last()]" mode="insertConnector">
 <xsl:call-template name="Copy" />

 <xsl:call-template name="AddConnector" />
 </xsl:template>

 <!-- ... or before the first Engine if there is no existing Connector -->
 <xsl:template match="Engine[1][not(preceding-sibling::Connector)]"
 mode="insertConnector">
 <xsl:call-template name="AddConnector" />

 <xsl:call-template name="Copy" />
 </xsl:template>

 <xsl:template name="AddConnector">
 <!-- Add new line -->
 <xsl:text>
</xsl:text>
 <!-- This is the new connector -->
 <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
 maxThreads="150" scheme="https" secure="true"
 keystoreFile="${user.home}/.keystore" keystorePass="changeit"
 clientAuth="false" sslProtocol="TLS" />
 </xsl:template>
 </xsl:template>
</xsl:stylesheet>

```

#### Finalize configuration

Finally, place the driver JARs in the Tomcat classpath and restart your App Service.

1. Ensure that the JDBC driver files are available to the Tomcat classloader by placing them in the `/home/tomcat/lib` directory. (Create this directory if it does not already exist.) To upload these files to your App Service instance, perform the following steps:

- a. In the [Cloud Shell](#), install the webapp extension:

```
az extension add --name webapp
```

- b. Run the following CLI command to create an SSH tunnel from your local system to App Service:

```
az webapp remote-connection create --resource-group <resource-group-name> --name <app-name> --port <port-on-local-machine>
```

- c. Connect to the local tunneling port with your SFTP client and upload the files to the `/home/tomcat/lib` folder.

Alternatively, you can use an FTP client to upload the JDBC driver. Follow these [instructions for getting your FTP credentials](#).

2. If you created a server-level data source, restart the App Service Linux application. Tomcat will reset `CATALINA_BASE` to `/home/tomcat` and use the updated configuration.

## JBoss EAP

There are three core steps when [registering a data source with JBoss EAP](#): uploading the JDBC driver, adding the JDBC driver as a module, and registering the module. App Service is a stateless hosting service, so the configuration commands for adding and registering the data source module must be scripted and applied as the container starts.

1. Obtain your database's JDBC driver.
2. Create an XML module definition file for the JDBC driver. The example shown below is a module definition for PostgreSQL.

```
<?xml version="1.0" ?>
<module xmlns="urn:jboss:module:1.1" name="org.postgres">
 <resources>
 <!-- ***** IMPORTANT : REPLACE THIS PLACEHOLDER *****-->
 <resource-root path="/home/site/deployments/tools/postgresql-42.2.12.jar" />
 </resources>
 <dependencies>
 <module name="javax.api"/>
 <module name="javax.transaction.api"/>
 </dependencies>
</module>
```

3. Put your JBoss CLI commands into a file named `jboss-cli-commands.cli`. The JBoss commands must add the module and register it as a data source. The example below shows the JBoss CLI commands for PostgreSQL.

```

#!/usr/bin/env bash
module add --name=org.postgres --resources=/home/site/deployments/tools/postgresql-42.2.12.jar --
module-xml=/home/site/deployments/tools/postgres-module.xml

/subsystem=datasources/jdbc-driver=postgres:add(driver-name="postgres",driver-module-
name="org.postgres",driver-class-name=org.postgresql.Driver,driver-xa-datasource-class-
name=org.postgresql.xa.PGXADatasource)

data-source add --name=postgresDS --driver-name=postgres --jndi-
name=java:jboss/datasources/postgresDS --connection-
url=${POSTGRES_CONNECTION_URL,env.POSTGRES_CONNECTION_URL:jdbc:postgresql://db:5432/postgres} --user-
name=${POSTGRES_SERVER_ADMIN_FULL_NAME,env.POSTGRES_SERVER_ADMIN_FULL_NAME:postgres} --
password=${POSTGRES_SERVER_ADMIN_PASSWORD,env.POSTGRES_SERVER_ADMIN_PASSWORD:example} --use-ccm=true
--max-pool-size=5 --blocking-timeout-wait-millis=5000 --enabled=true --driver-
class=org.postgresql.Driver --exception-sorter-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLExceptionSorter --jta=true --use-java-
context=true --valid-connection-checker-class-
name=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker

```

4. Create a startup script, `startup_script.sh` that calls the JBoss CLI commands. The example below shows how to call your `jboss-cli-commands.cli`. Later you will configure App Service to run this script when the container starts.

```
$JBOSS_HOME/bin/jboss-cli.sh --connect --file=/home/site/deployments/tools/jboss-cli-commands.cli
```

5. Using an FTP client of your choice, upload your JDBC driver, `jboss-cli-commands.cli`, `startup_script.sh`, and the module definition to `/site/deployments/tools/`.
6. Configure your site to run `startup_script.sh` when the container starts. In the Azure portal, navigate to **Configuration > General Settings > Startup Command**. Set the startup command field to `/home/site/deployments/tools/startup_script.sh`. Save your changes.

To confirm that the datasource was added to the JBoss server, SSH into your webapp and run `$JBOSS_HOME/bin/jboss-cli.sh --connect`. Once you are connected to JBoss run the `/subsystem=datasources:read-resource` to print a list of the data sources.

## robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "--" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415
"_" "_"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

## Choosing a Java runtime version

App Service allows users to choose the major version of the JVM, such as Java 8 or Java 11, and the minor version, such as 1.8.0\_232 or 11.0.5. You can also choose to have the minor version automatically updated as new minor versions become available. In most cases, production sites should use pinned minor JVM versions. This will prevent unanticipated outages during a minor version auto-update. All Java web apps use 64-bit JVMs, this is not configurable.

If you choose to pin the minor version, you will need to periodically update the JVM minor version on the site. To

ensure that your application runs on the newer minor version, create a staging slot and increment the minor version on the staging site. Once you have confirmed the application runs correctly on the new minor version, you can swap the staging and production slots.

## JBoss EAP App Service Plans

JBoss EAP is only available on the Premium v3 and Isolated v2 App Service Plan types. Customers that created a JBoss EAP site on a different tier during the public preview should scale up to Premium or Isolated hardware tier to avoid unexpected behavior.

## Java runtime statement of support

### JDK versions and maintenance

Azure's supported Java Development Kit (JDK) is [Zulu](#) provided through [Azul Systems](#). Azul Zulu Enterprise builds of OpenJDK are a no-cost, multi-platform, production-ready distribution of the OpenJDK for Azure and Azure Stack backed by Microsoft and Azul Systems. They contain all the components for building and running Java SE applications. You can install the JDK from [Java JDK Installation](#).

Major version updates will be provided through new runtime options in Azure App Service. Customers update to these newer versions of Java by configuring their App Service deployment and are responsible for testing and ensuring the major update meets their needs.

Supported JDKs are automatically patched on a quarterly basis in January, April, July, and October of each year. For more information on Java on Azure, see [this support document](#).

### Security updates

Patches and fixes for major security vulnerabilities will be released as soon as they become available from Azul Systems. A "major" vulnerability is defined by a base score of 9.0 or higher on the [NIST Common Vulnerability Scoring System, version 2](#).

Tomcat 8.0 has reached [End of Life \(EOL\) as of September 30, 2018](#). While the runtime is still available on Azure App Service, Azure will not apply security updates to Tomcat 8.0. If possible, migrate your applications to Tomcat 8.5 or 9.0. Both Tomcat 8.5 and 9.0 are available on Azure App Service. See the [official Tomcat site](#) for more information.

### Deprecation and retirement

If a supported Java runtime will be retired, Azure developers using the affected runtime will be given a deprecation notice at least six months before the runtime is retired.

### Local development

Developers can download the Production Edition of Azul Zulu Enterprise JDK for local development from [Azul's download site](#).

### Development support

Product support for the [Azure-supported Azul Zulu JDK](#) is available through Microsoft when developing for Azure or [Azure Stack](#) with a [qualified Azure support plan](#).

## Next steps

Visit the [Azure for Java Developers](#) center to find Azure quickstarts, tutorials, and Java reference documentation.

- [App Service Linux FAQ](#)
- [Environment variables and app settings reference](#)

# Configure a Linux Ruby app for Azure App Service

11/2/2021 • 5 minutes to read • [Edit Online](#)

This article describes how [Azure App Service](#) runs Ruby apps in a Linux container, and how you can customize the behavior of App Service when needed. Ruby apps must be deployed with all the required [gems](#).

This guide provides key concepts and instructions for Ruby developers who use a built-in Linux container in App Service. If you've never used Azure App Service, you should follow the [Ruby quickstart](#) and [Ruby with PostgreSQL tutorial](#) first.

## Show Ruby version

To show the current Ruby version, run the following command in the [Cloud Shell](#):

```
az webapp config show --resource-group <resource-group-name> --name <app-name> --query linuxFxVersion
```

To show all supported Ruby versions, run the following command in the [Cloud Shell](#):

```
az webapp list-runtimes --linux | grep RUBY
```

You can run an unsupported version of Ruby by building your own container image instead. For more information, see [use a custom Docker image](#).

## Set Ruby version

Run the following command in the [Cloud Shell](#) to set the Ruby version to 2.3:

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "RUBY|2.3"
```

### NOTE

If you see errors similar to the following during deployment time:

```
Your Ruby version is 2.3.3, but your Gemfile specified 2.3.1
```

or

```
rbenv: version `2.3.1' is not installed
```

It means that the Ruby version configured in your project is different than the version that's installed in the container you're running (2.3.3 in the example above). In the example above, check both *Gemfile* and *.ruby-version* and verify that the Ruby version is not set, or is set to the version that's installed in the container you're running (2.3.3 in the example above).

## Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them using the standard `ENV['<path-name>']` pattern. For example, to access an app setting called WEBSITE\_SITE\_NAME, use the following

code:

```
ENV['WEBSITE_SITE_NAME']
```

## Customize deployment

When you deploy a [Git repository](#), or a [Zip package with build automation enabled](#), the deployment engine (Kudu) automatically runs the following post-deployment steps by default:

1. Check if a *Gemfile* exists.
2. Run `bundle clean`.
3. Run `bundle install --path "vendor/bundle"`.
4. Run `bundle package` to package gems into vendor/cache folder.

### Use `--without` flag

To run `bundle install` with the `--without` flag, set the `BUNDLE_WITHOUT` app setting to a comma-separated list of groups. For example, the following command sets it to `development,test`.

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
BUNDLE_WITHOUT="development,test"
```

If this setting is defined, then the deployment engine runs `bundle install` with `--without $BUNDLE_WITHOUT`.

### Precompile assets

The post-deployment steps don't precompile assets by default. To turn on asset precompilation, set the `ASSETS_PRECOMPILE` app setting to `true`. Then the command `bundle exec rake --trace assets:precompile` is run at the end of the post-deployment steps. For example:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
ASSETS_PRECOMPILE=true
```

For more information, see [Serve static assets](#).

## Customize start-up

By default, the Ruby container starts the Rails server in the following sequence (for more information, see the [start-up script](#)):

1. Generate a `secret_key_base` value, if one doesn't exist already. This value is required for the app to run in production mode.
2. Set the `RAILS_ENV` environment variable to `production`.
3. Delete any `.pid` file in the `tmp/pids` directory that's left by a previously running Rails server.
4. Check if all dependencies are installed. If not, try installing gems from the local `vendor/cache` directory.
5. Run `rails server -e $RAILS_ENV`.

You can customize the start-up process in the following ways:

- [Serve static assets](#)
- [Run in non-production mode](#)
- [Set `secret\_key\_base` manually](#)

### Serve static assets

The Rails server in the Ruby container runs in production mode by default, and [assumes that assets are precompiled and are served by your web server](#). To serve static assets from the Rails server, you need to do two things:

- **Precompile the assets** - [Precompile the static assets locally](#) and deploy them manually. Or, let the deployment engine handle it instead (see [Precompile assets](#)).
- **Enable serving static files** - To serve static assets from the Ruby container, [set the RAILS\\_SERVE\\_STATIC\\_FILES app setting](#) to `true`. For example:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
RAILS_SERVE_STATIC_FILES=true
```

## Run in non-production mode

The Rails server runs in production mode by default. To run in development mode, for example, set the [RAILS\\_ENV app setting](#) to `development`.

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
RAILS_ENV="development"
```

However, this setting alone causes the Rails server to start in development mode, which accepts localhost requests only and isn't accessible outside of the container. To accept remote client requests, set the [APP\\_COMMAND\\_LINE app setting](#) to `rails server -b 0.0.0.0`. This app setting lets you run a custom command in the Ruby container. For example:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
APP_COMMAND_LINE="rails server -b 0.0.0.0"
```

## Set secret\_key\_base manually

To use your own [secret\\_key\\_base](#) value instead of letting App Service generate one for you, set the [SECRET\\_KEY\\_BASE app setting](#) with the value you want. For example:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
SECRET_KEY_BASE=<key-base-value>
```

## Access diagnostic logs

To access the console logs generated from inside your application code in App Service, turn on diagnostics logging by running the following command in the [Cloud Shell](#):

```
az webapp log config --resource-group <resource-group-name> --name <app-name> --docker-container-logging
filesystem --level Verbose
```

Possible values for `--level` are: `Error`, `Warning`, `Info`, and `Verbose`. Each subsequent level includes the previous level. For example: `Error` includes only error messages, and `Verbose` includes all messages.

Once diagnostic logging is turned on, run the following command to see the log stream:

```
az webapp log tail --resource-group <resource-group-name> --name <app-name>
```

If you don't see console logs immediately, check again in 30 seconds.

#### NOTE

You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

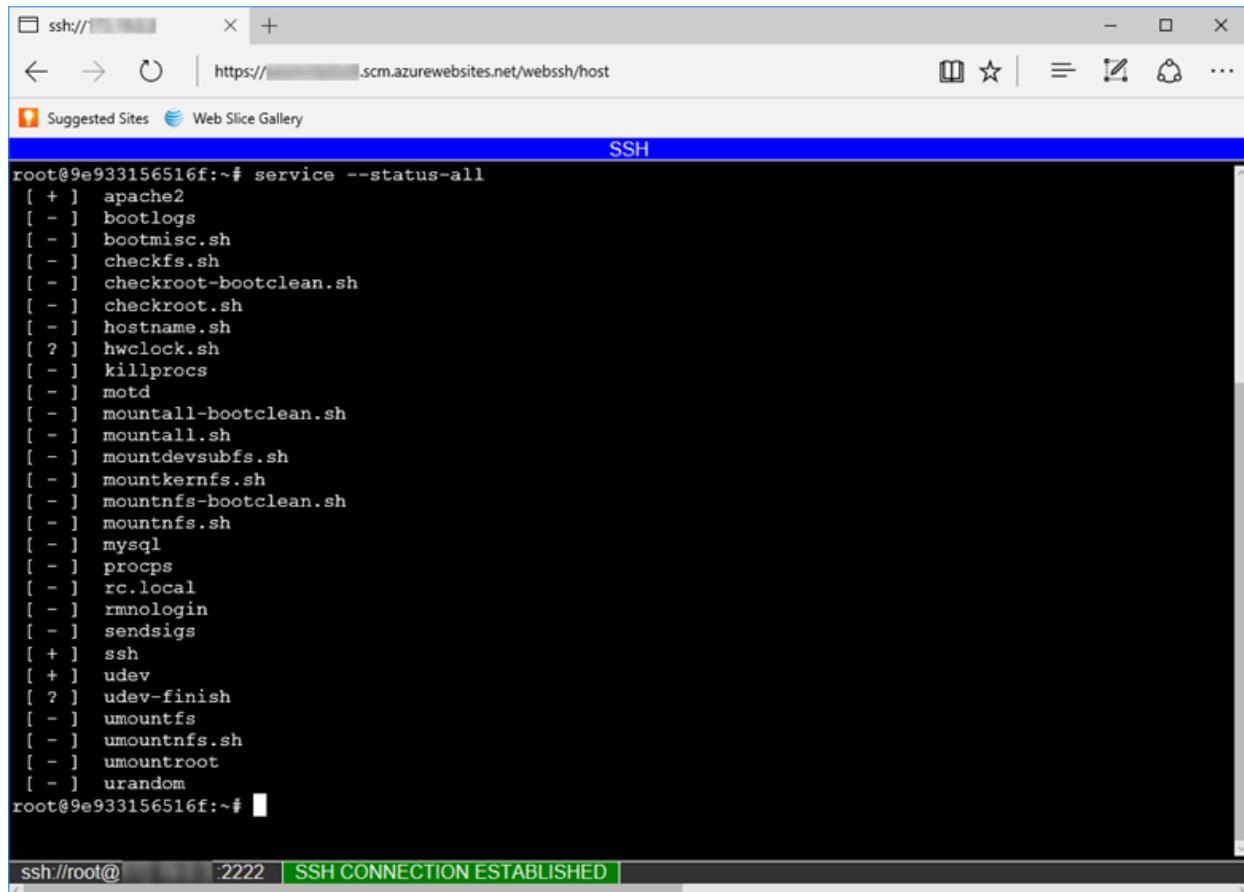
## Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

```
https://<app-name>.scm.azurewebsites.net/webssh/host
```

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.



#### NOTE

Any changes you make outside the `/home` directory are stored in the container itself and don't persist beyond an app restart.

To open a remote SSH session from your local machine, see [Open SSH session from remote shell](#).

## robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415
"_" "_"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

## More resources

- [Tutorial: Rails app with PostgreSQL](#)
- [App Service Linux FAQ](#)
- [Environment variables and app settings reference](#)

# Configure a custom container for Azure App Service

11/2/2021 • 17 minutes to read • [Edit Online](#)

This article shows you how to configure a custom container to run on Azure App Service.

This guide provides key concepts and instructions for containerization of Windows apps in App Service. If you've never used Azure App Service, follow the [custom container quickstart](#) and [tutorial](#) first.

This guide provides key concepts and instructions for containerization of Linux apps in App Service. If you've never used Azure App Service, follow the [custom container quickstart](#) and [tutorial](#) first. There's also a [multi-container app quickstart](#) and [tutorial](#).

## Supported parent images

For your custom Windows image, you must choose the right [parent image \(base image\)](#) for the framework you want:

- To deploy .NET Framework apps, use a parent image based on the Windows Server 2019 Core [Long-Term Servicing Channel \(LTSC\)](#) release.
- To deploy .NET Core apps, use a parent image based on the Windows Server 2019 Nano [Semi-Annual Servicing Channel \(SAC\)](#) release.

It takes some time to download a parent image during app start-up. However, you can reduce start-up time by using one of the following parent images that are already cached in Azure App Service:

- [mcr.microsoft.com/windows/servercore:20H2](https://mcr.microsoft.com/windows/servercore:20H2)
- [mcr.microsoft.com/windows/servercore:ltsc2019](https://mcr.microsoft.com/windows/servercore:ltsc2019)
- [mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-20H2](https://mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-20H2)
- [mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-ltsc2019](https://mcr.microsoft.com/dotnet/framework/aspnet:4.8-windowsservercore-ltsc2019)
- [mcr.microsoft.com/dotnet/runtime:5.0-nanoserver-20H2](https://mcr.microsoft.com/dotnet/runtime:5.0-nanoserver-20H2)
- [mcr.microsoft.com/dotnet/runtime:5.0-nanoserver-1809](https://mcr.microsoft.com/dotnet/runtime:5.0-nanoserver-1809)
- [mcr.microsoft.com/dotnet/aspnet:5.0-nanoserver-20H2](https://mcr.microsoft.com/dotnet/aspnet:5.0-nanoserver-20H2)
- [mcr.microsoft.com/dotnet/aspnet:5.0-nanoserver-1809](https://mcr.microsoft.com/dotnet/aspnet:5.0-nanoserver-1809)
- [mcr.microsoft.com/dotnet/runtime:3.1-nanoserver-20H2](https://mcr.microsoft.com/dotnet/runtime:3.1-nanoserver-20H2)
- [mcr.microsoft.com/dotnet/runtime:3.1-nanoserver-1809](https://mcr.microsoft.com/dotnet/runtime:3.1-nanoserver-1809)
- [mcr.microsoft.com/dotnet/aspnet:3.1-nanoserver-20H2](https://mcr.microsoft.com/dotnet/aspnet:3.1-nanoserver-20H2)
- [mcr.microsoft.com/dotnet/aspnet:3.1-nanoserver-1809](https://mcr.microsoft.com/dotnet/aspnet:3.1-nanoserver-1809)

## Change the Docker image of a custom container

To change an existing custom container app from the current Docker image to a new image, use the following command:

```
az webapp config container set --name <app-name> --resource-group <group-name> --docker-custom-image-name <docker-hub-repo>/<image>
```

## Use an image from a private registry

To use an image from a private registry, such as Azure Container Registry, run the following command:

```
az webapp config container set --name <app-name> --resource-group <group-name> --docker-custom-image-name <image-name> --docker-registry-server-url <private-repo-url> --docker-registry-server-user <username> --docker-registry-server-password <password>
```

For *<username>* and *<password>*, supply the login credentials for your private registry account.

## Use managed identity to pull image from Azure Container Registry

Use the following steps to configure your web app to pull from ACR using managed identity. The steps will use system-assigned managed identity, but you can use user-assigned managed identity as well.

1. Enable [the system-assigned managed identity](#) for the web app by using the `az webapp identity assign` command:

```
az webapp identity assign --resource-group <group-name> --name <app-name> --query principalId --output tsv
```

Replace *<app-name>* with the name you used in the previous step. The output of the command (filtered by the `--query` and `--output` arguments) is the service principal ID of the assigned identity, which you use shortly.

2. Get the resource ID of your Azure Container Registry:

```
az acr show --resource-group <group-name> --name <registry-name> --query id --output tsv
```

Replace *<registry-name>* with the name of your registry. The output of the command (filtered by the `--query` and `--output` arguments) is the resource ID of the Azure Container Registry.

3. Grant the managed identity permission to access the container registry:

```
az role assignment create --assignee <principal-id> --scope <registry-resource-id> --role "AcrPull"
```

Replace the following values:

- *<principal-id>* with the service principal ID from the `az webapp identity assign` command
- *<registry-resource-id>* with the ID of your container registry from the `az acr show` command

For more information about these permissions, see [What is Azure role-based access control](#).

4. Configure your app to use the managed identity to pull from Azure Container Registry.

```
az webapp config set --resource-group <group-name> --name <app-name> --generic-configurations '{"acrUseManagedIdentityCreds": true}'
```

Replace the following values:

- *<app-name>* with the name of your web app.

### TIP

If you are using PowerShell console to run the commands, you will need to escape the strings in the `--generic-configurations` argument in this and the next step. For example:

```
--generic-configurations '{"acrUseManagedIdentityCreds": true}'
```

5. (Optional) If your app uses a [user-assigned managed identity](#), make sure this is configured on the web app and then set an additional `acrUserManagedIdentityID` property to specify its client ID:

```
az identity show --resource-group <group-name> --name <identity-name> --query clientId --output tsv
```

Replace the `<identity-name>` of your user-assigned managed identity and use the output `<client-id>` to configure the user-assigned managed identity ID.

```
az webapp config set --resource-group <group-name> --name <app-name> --generic-configurations '{"acrUserManagedIdentityID": "<client-id>"}'
```

You are all set, and the web app will now use managed identity to pull from Azure Container Registry.

## Use an image from a network protected registry

To connect and pull from a registry inside a virtual network or on-premises, your app will need to be connected to a virtual network using the VNet integration feature. This is also need for Azure Container Registry with private endpoint. When your network and DNS resolution is configured, you enable the routing of the image pull through the VNet by setting the App Setting `WEBSITE_PULL_IMAGE_OVER_VNET=true`:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings WEBSITE_PULL_IMAGE_OVER_VNET=true
```

## I don't see the updated container

If you change your Docker container settings to point to a new container, it may take a few minutes before the app serves HTTP requests from the new container. While the new container is being pulled and started, App Service continues to serve requests from the old container. Only when the new container is started and ready to receive requests does App Service start sending requests to it.

## How container images are stored

The first time you run a custom Docker image in App Service, App Service does a `docker pull` and pulls all image layers. These layers are stored on disk, like if you were using Docker on-premises. Each time the app restarts, App Service does a `docker pull`, but only pulls layers that have changed. If there have been no changes, App Service uses existing layers on the local disk.

If the app changes compute instances for any reason, such as scaling up and down the pricing tiers, App Service must pull down all layers again. The same is true if you scale out to add additional instances. There are also rare cases where the app instances may change without a scale operation.

## Configure port number

By default, App Service assumes your custom container is listening on port 80. If your container listens to a different port, set the `WEBSITES_PORT` app setting in your App Service app. You can set it via the [Cloud Shell](#). In

Bash:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings
WEBSITES_PORT=8000
```

In PowerShell:

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings @{"WEBSITES_PORT"="8000"}
```

App Service currently allows your container to expose only one port for HTTP requests.

## Configure environment variables

Your custom container may use environment variables that need to be supplied externally. You can pass them in via the [Cloud Shell](#). In Bash:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings
DB_HOST="myownserver.mysql.database.azure.com"
```

In PowerShell:

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings
@{ "DB_HOST"="myownserver.mysql.database.azure.com"}
```

When your app runs, the App Service app settings are injected into the process as environment variables automatically. You can verify container environment variables with the URL

<https://<app-name>.scm.azurewebsites.net/Env>.

If your app uses images from a private registry or from Docker Hub, credentials for accessing the repository are saved in environment variables: `DOCKER_REGISTRY_SERVER_URL`, `DOCKER_REGISTRY_SERVER_USERNAME` and `DOCKER_REGISTRY_SERVER_PASSWORD`. Because of security risks, none of these reserved variable names are exposed to the application.

For IIS or .NET Framework (4.0 or above) based containers, they're injected into `System.ConfigurationManager` as .NET app settings and connection strings automatically by App Service. For all other language or framework, they're provided as environment variables for the process, with one of the following corresponding prefixes:

- `APPSETTING_`
- `SQLCONTR_`
- `MYSQLCONTR_`
- `SQLAZURECOSTR_`
- `POSTGRESQLCONTR_`
- `CUSTOMCONNSTR_`

This method works both for single-container apps or multi-container apps, where the environment variables are specified in the `docker-compose.yml` file.

## Use persistent shared storage

You can use the `C:\home` directory in your app's file system to persist files across restarts and share them across instances. The `c:\home` in your app is provided to enable your container app to access persistent storage.

When persistent storage is disabled, writes to the `C:\home` directory aren't persisted. [Docker host logs and container logs](#) are saved in a default persistent shared storage that is not attached to the container. When persistent storage is enabled, all writes to the `C:\home` directory are persisted and can be accessed by all instances of a scaled-out app, and log are accessible at `C:\home\LogFiles`.

You can use the `/home` directory in your app's file system to persist files across restarts and share them across instances. The `/home` in your app is provided to enable your container app to access persistent storage.

When persistent storage is disabled, then writes to the `/home` directory aren't persisted across app restarts or across multiple instances. The only exception is the `/home/LogFiles` directory, which is used to store the Docker and container logs. When persistent storage is enabled, all writes to the `/home` directory are persisted and can be accessed by all instances of a scaled-out app.

By default, persistent storage is disabled and the setting is not exposed in the app settings. To enable it, set the `WEBSITES_ENABLE_APP_SERVICE_STORAGE` app setting via the [Cloud Shell](#). In Bash:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings
WEBSITES_ENABLE_APP_SERVICE_STORAGE=true
```

In PowerShell:

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings
@{"WEBSITES_ENABLE_APP_SERVICE_STORAGE"=true}
```

#### NOTE

You can also [configure your own persistent storage](#).

## Detect HTTPS session

App Service terminates TLS/SSL at the front ends. That means that TLS/SSL requests never get to your app. You don't need to, and shouldn't implement any support for TLS/SSL into your app.

The front ends are located inside Azure data centers. If you use TLS/SSL with your app, your traffic across the Internet will always be safely encrypted.

## Customize ASP.NET machine key injection

During the container start, automatically generated keys are injected into the container as the machine keys for ASP.NET cryptographic routines. You can [find these keys in your container](#) by looking for the following environment variables: `MACHINEKEY_Decryption`, `MACHINEKEY_DecryptionKey`, `MACHINEKEY_ValidationKey`, `MACHINEKEY_Validation`.

The new keys at each restart may reset ASP.NET forms authentication and view state, if your app depends on them. To prevent the automatic regeneration of keys, [set them manually as App Service app settings](#).

## Connect to the container

You can connect to your Windows container directly for diagnostic tasks by navigating to

`https://<app-name>.scm.azurewebsites.net/DebugConsole`. Here's how it works:

- The debug console lets you execute interactive commands, such as starting PowerShell sessions, inspecting registry keys, and navigate the entire container file system.

- It functions separately from the graphical browser above it, which only shows the files in your [shared storage](#).
- In a scaled-out app, the debug console is connected to one of the container instances. You can select a different instance from the **Instance** dropdown in the top menu.
- Any change you make to the container from within the console does *not* persist when your app is restarted (except for changes in the shared storage), because it's not part of the Docker image. To persist your changes, such as registry settings and software installation, make them part of the Dockerfile.

## Access diagnostic logs

App Service logs actions by the Docker host as well as activities from within the container. Logs from the Docker host (platform logs) are shipped by default, but application logs or web server logs from within the container need to be enabled manually. For more information, see [Enable application logging](#) and [Enable web server logging](#).

There are several ways to access Docker logs:

- [In Azure portal](#)
- [From the Kudu console](#)
- [With the Kudu API](#)
- [Send logs to Azure monitor](#)

### In Azure portal

Docker logs are displayed in the portal, in the **Container Settings** page of your app. The logs are truncated, but you can download all the logs clicking **Download**.

### From the Kudu console

Navigate to <https://<app-name>.scm.azurewebsites.net/DebugConsole> and click the **LogFiles** folder to see the individual log files. To download the entire **LogFiles** directory, click the **Download** icon to the left of the directory name. You can also access this folder using an FTP client.

In the console terminal, you can't access the `c:\home\LogFiles` folder by default because persistent shared storage is not enabled. To enable this behavior in the console terminal, [enable persistent shared storage](#).

If you try to download the Docker log that is currently in use using an FTP client, you may get an error because of a file lock.

### With the Kudu API

Navigate directly to <https://<app-name>.scm.azurewebsites.net/api/logs/docker> to see metadata for the Docker logs. You may see more than one log file listed, and the `href` property lets you download the log file directly.

To download all the logs together in one ZIP file, access

<https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip>.

## Customize container memory

By default all Windows Containers deployed in Azure App Service are limited to 1 GB RAM. You can change this value by providing the `WEBSITE_MEMORY_LIMIT_MB` app setting via the [Cloud Shell](#). In Bash:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings
WEBSITE_MEMORY_LIMIT_MB=2000
```

In PowerShell:

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings @{"WEBSITE_MEMORY_LIMIT_MB"=2000}
```

The value is defined in MB and must be less and equal to the total physical memory of the host. For example, in an App Service plan with 8 GB RAM, the cumulative total of `WEBSITE_MEMORY_LIMIT_MB` for all the apps must not exceed 8 GB. Information on how much memory is available for each pricing tier can be found in [App Service pricing](#), in the [Premium v3 service plan](#) section.

## Customize the number of compute cores

By default, a Windows container runs with all available cores for your chosen pricing tier. You may want to reduce the number of cores that your staging slot uses, for example. To reduce the number of cores used by a container, set the `WEBSITE_CPU_CORES_LIMIT` app setting to the preferred number of cores. You can set it via the [Cloud Shell](#). In Bash:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --slot staging --settings WEBSITE_CPU_CORES_LIMIT=1
```

In PowerShell:

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings @{"WEBSITE_CPU_CORES_LIMIT"=1}
```

### NOTE

Updating the app setting triggers automatic restart, causing minimal downtime. For a production app, consider swapping it into a staging slot, change the app setting in the staging slot, and then swap it back into production.

Verify your adjusted number by going to the Kudu Console (<https://<app-name>.scm.azurewebsites.net>) and typing in the following commands using PowerShell. Each command outputs a number.

```
Get-ComputerInfo | ft CsNumberOfLogicalProcessors # Total number of enabled logical processors. Disabled processors are excluded.
Get-ComputerInfo | ft CsNumberOfProcessors # Number of physical processors.
```

The processors may be multicore or hyperthreading processors. Information on how many cores are available for each pricing tier can be found in [App Service pricing](#), in the [Premium v3 service plan](#) section.

## Customize health ping behavior

App Service considers a container to be successfully started when the container starts and responds to an HTTP ping. The health ping request contains the header

`User-Agent= "App Service Hyper-V Container Availability Check"`. If the container starts but does not respond to a ping after a certain amount of time, App Service logs an event in the Docker log, saying that the container didn't start.

If your application is resource-intensive, the container might not respond to the HTTP ping in time. To control the actions when HTTP pings fail, set the `CONTAINER_AVAILABILITY_CHECK_MODE` app setting. You can set it via the [Cloud Shell](#). In Bash:

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings CONTAINER_AVAILABILITY_CHECK_MODE="ReportOnly"
```

In PowerShell:

```
Set-AzWebApp -ResourceGroupName <group-name> -Name <app-name> -AppSettings
@{"CONTAINER_AVAILABILITY_CHECK_MODE"="ReportOnly"}
```

The following table shows the possible values:

VALUE	DESCRIPTIONS
Repair	Restart the container after three consecutive availability checks
ReportOnly	The default value. Don't restart the container but report in the Docker logs for the container after three consecutive availability checks.
Off	Don't check for availability.

## Support for Group Managed Service Accounts

Group Managed Service Accounts (gMSAs) are currently not supported in Windows containers in App Service.

## Enable SSH

SSH enables secure communication between a container and a client. In order for a custom container to support SSH, you must add it into your Docker image itself.

### TIP

All built-in Linux containers in App Service have added the SSH instructions in their image repositories. You can go through the following instructions with the [Node.js 10.14 repository](#) to see how it's enabled there. The configuration in the Node.js built-in image is slightly different, but the same in principle.

- Add [an sshd\\_config file](#) to your repository, like the following example.

```
Port 2222
ListenAddress 0.0.0.0
LoginGraceTime 180
X11Forwarding yes
Ciphers aes128-cbc,3des-cbc,aes256-cbc,aes128-ctr,aes192-ctr,aes256-ctr
MACs hmac-sha1,hmac-sha1-96
StrictModes yes
SyslogFacility DAEMON
PasswordAuthentication yes
PermitEmptyPasswords no
PermitRootLogin yes
Subsystem sftp internal-sftp
```

## NOTE

This file configures OpenSSH and must include the following items:

- **Port** must be set to 2222.
- **Ciphers** must include at least one item in this list: `aes128-cbc,3des-cbc,aes256-cbc`.
- **MACs** must include at least one item in this list: `hmac-sha1,hmac-sha1-96`.

- Add an `ssh_setup` script file to create the SSH keys [using ssh-keygen](#) to your repository.

```
#!/bin/sh

if [! -f "/etc/ssh/ssh_host_rsa_key"]; then
 # generate fresh rsa key
 ssh-keygen -f /etc/ssh/ssh_host_rsa_key -N '' -t rsa
fi

if [! -f "/etc/ssh/ssh_host_dsa_key"]; then
 # generate fresh dsa key
 ssh-keygen -f /etc/ssh/ssh_host_dsa_key -N '' -t dsa
fi

if [! -f "/etc/ssh/ssh_host_ecdsa_key"]; then
 # generate fresh ecdsa key
 ssh-keygen -f /etc/ssh/ssh_host_ecdsa_key -N '' -t dsa
fi

if [! -f "/etc/ssh/ssh_host_ed25519_key"]; then
 # generate fresh ed25519 key
 ssh-keygen -f /etc/ssh/ssh_host_ed25519_key -N '' -t dsa
fi

#prepare run dir
if [! -d "/var/run/sshd"]; then
 mkdir -p /var/run/sshd
fi
```

- In your Dockerfile, add the following commands:

```
Install OpenSSH and set the password for root to "Docker!". In this example, "apk add" is the
install instruction for an Alpine Linux-based image.
RUN apk add openssh \
 && echo "root:Docker!" | chpasswd

Copy the sshd_config file to the /etc/ssh/ directory
COPY sshd_config /etc/ssh/

Copy and configure the ssh_setup file
RUN mkdir -p /tmp
COPY ssh_setup.sh /tmp
RUN chmod +x /tmp/ssh_setup.sh \
 && (sleep 1;/tmp/ssh_setup.sh 2>&1 > /dev/null)

Open port 2222 for SSH access
EXPOSE 80 2222
```

#### NOTE

The root password must be exactly `Docker!` as it is used by App Service to let you access the SSH session with the container. This configuration doesn't allow external connections to the container. Port 2222 of the container is accessible only within the bridge network of a private virtual network and is not accessible to an attacker on the internet.

- In the start-up script for your container, start the SSH server.

```
/usr/sbin/sshd
```

## Access diagnostic logs

You can access the console logs generated from inside the container.

First, turn on container logging by running the following command:

```
az webapp log config --name <app-name> --resource-group <resource-group-name> --docker-container-logging filesystem
```

Replace `<app-name>` and `<resource-group-name>` with the names appropriate for your web app.

Once container logging is turned on, run the following command to see the log stream:

```
az webapp log tail --name <app-name> --resource-group <resource-group-name>
```

If you don't see console logs immediately, check again in 30 seconds.

To stop log streaming at any time, type **Ctrl+C**.

You can also inspect the log files in a browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

## Configure multi-container apps

- [Use persistent storage in Docker Compose](#)
- [Preview limitations](#)
- [Docker Compose options](#)

### Use persistent storage in Docker Compose

Multi-container apps like WordPress need persistent storage to function properly. To enable it, your Docker Compose configuration must point to a storage location *outside* your container. Storage locations inside your container don't persist changes beyond app restart.

Enable persistent storage by setting the `WEBSITES_ENABLE_APP_SERVICE_STORAGE` app setting, using the [az webapp config appsettings set](#) command in [Cloud Shell](#).

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings WEBSITES_ENABLE_APP_SERVICE_STORAGE=TRUE
```

In your `docker-compose.yml` file, map the `volumes` option to  `${WEBAPP_STORAGE_HOME}` .

`WEBAPP_STORAGE_HOME` is an environment variable in App Service that is mapped to persistent storage for your app. For example:

```
wordpress:
 image: <image name:tag>
 volumes:
 - ${WEBAPP_STORAGE_HOME}/site/wwwroot:/var/www/html
 - ${WEBAPP_STORAGE_HOME}/phpmyadmin:/var/www/phpmyadmin
 - ${WEBAPP_STORAGE_HOME}/LogFiles:/var/log
```

## Preview limitations

Multi-container is currently in preview. The following App Service platform features are not supported:

- Authentication / Authorization
- Managed Identities
- CORS
- VNET integration is not supported for Docker Compose scenarios
- Docker Compose on Azure App Services currently has a limit of 4,000 characters at this time.

## Docker Compose options

The following lists show supported and unsupported Docker Compose configuration options:

### Supported options

- command
- entrypoint
- environment
- image
- ports
- restart
- services
- volumes

### Unsupported options

- build (not allowed)
- [depends\\_on](#) (ignored)
- networks (ignored)
- secrets (ignored)
- ports other than 80 and 8080 (ignored)

### NOTE

Any other options not explicitly called out are ignored in Public Preview.

## robots933456 in logs

You may see the following message in the container logs:

```
2019-04-08T14:07:56.641002476Z "-" - - [08/Apr/2019:14:07:56 +0000] "GET /robots933456.txt HTTP/1.1" 404 415
"-_-_-_-"
```

You can safely ignore this message. `/robots933456.txt` is a dummy URL path that App Service uses to check if the container is capable of serving requests. A 404 response simply indicates that the path doesn't exist, but it lets App Service know that the container is healthy and ready to respond to requests.

## Next steps

[Tutorial: Migrate custom software to Azure App Service using a custom container](#)

[Tutorial: Multi-container WordPress app](#)

Or, see additional resources:

- [Environment variables and app settings reference](#)
- [Load certificate in Windows/Linux containers](#)

# Mount Azure Storage as a local share in a container app in App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

## NOTE

Azure Storage in App Service Windows container is in [preview](#) and **not supported for production scenarios**.

This guide shows how to mount Azure Storage Files as a network share in a Windows container in App Service. Only [Azure Files Shares](#) and [Premium Files Shares](#) are supported. The benefits of custom-mounted storage include:

This guide shows how to mount Azure Storage as a network share in a built-in Linux container or a custom Linux container in App Service. The benefits of custom-mounted storage include:

- Configure persistent storage for your App Service app and manage the storage separately.
- Make static content like video and images readily available for your App Service app.
- Write application log files or archive older application log to Azure File shares.
- Share content across multiple apps or with other Azure services.

The following features are supported for Windows containers:

- Secured access to storage accounts with [private links](#) (when [VNET integration](#) is used). [Service endpoint](#) support is currently unavailable.
- Azure Files (read/write).
- Up to five mount points per app.
- Drive letter assignments ( `c:` to `z:` ).

The following features are supported for Linux containers:

- Secured access to storage accounts with [service endpoints](#) and [private links](#) (when [VNET integration](#) is used).
- Azure Files (read/write).
- Azure Blobs (read-only).
- Up to five mount points per app.

## Prerequisites

- An existing Windows Container app in [Azure App Service](#)
- [Create Azure file share](#)
- [Upload files to Azure File share](#)
- An existing [App Service on Linux app](#).
- An [Azure Storage Account](#)
- An [Azure file share and directory](#).

## NOTE

Azure Storage is non-default storage for App Service and billed separately, not included with App Service.

## Limitations

- Storage mounts are not supported for native Windows (non-containerized) apps.
- Azure blobs are not supported.
- **Storage firewall** is supported only through [private endpoints](#) (when [VNET integration](#) is used). Custom DNS support is currently unavailable when the mounted Azure Storage account uses a private endpoint.
- FTP/FTPS access to mounted storage not supported (use [Azure Storage Explorer](#)).
- Mapping `[c-z]:\`, `[c-z]:\home`, `/`, and `/home` to custom-mounted storage is not supported.
- Storage mounts cannot be used together with clone settings option during [deployment slot](#) creation.
- Storage mounts are not backed up when you [back up your app](#). Be sure to follow best practices to back up the Azure Storage accounts.
- **Storage firewall** is supported only through [service endpoints](#) and [private endpoints](#) (when [VNET integration](#) is used). Custom DNS support is currently unavailable when the mounted Azure Storage account uses a private endpoint.
- FTP/FTPS access to custom-mounted storage is not supported (use [Azure Storage Explorer](#)).
- Azure CLI, Azure PowerShell, and Azure SDK support is in preview.
- Mapping `/` or `/home` to custom-mounted storage is not supported.
- Storage mounts cannot be used together with clone settings option during [deployment slot](#) creation.
- Storage mounts are not backed up when you [back up your app](#). Be sure to follow best practices to back up the Azure Storage accounts.

## Mount storage to Windows container

### Mount storage to Linux container

- [Azure portal](#)
- [Azure CLI](#)

1. In the [Azure portal](#), navigate to the app.
2. From the left navigation, click **Configuration > Path Mappings > New Azure Storage Mount**.
3. Configure the storage mount according to the following table. When finished, click **OK**.

SETTING	DESCRIPTION
Name	Name of the mount configuration. Spaces are not allowed.
Configuration options	Select <b>Basic</b> if the storage account is not using <a href="#">private endpoints</a> . Otherwise, select <b>Advanced</b> .
Storage accounts	Azure Storage account. It must contain an Azure Files share.
Share name	Files share to mount.
Access key (Advanced only)	<a href="#">Access key</a> for your storage account.

SETTING	DESCRIPTION
<b>Mount path</b>	Directory inside the Windows container to mount to Azure Storage. Do not use a root directory ([C-Z]:\ or /) or the home directory ([C-Z]:\home , or /home ).
SETTING	DESCRIPTION
<b>Name</b>	Name of the mount configuration. Spaces are not allowed.
<b>Configuration options</b>	Select <b>Basic</b> if the storage account is not using <a href="#">service endpoints</a> or <a href="#">private endpoints</a> . Otherwise, select <b>Advanced</b> .
<b>Storage accounts</b>	Azure Storage account.
<b>Storage type</b>	Select the type based on the storage you want to mount. Azure Blobs only supports read-only access.
<b>Storage container or Share name</b>	Files share or Blobs container to mount.
<b>Access key</b> (Advanced only)	<a href="#">Access key</a> for your storage account.
<b>Mount path</b>	Directory inside the Linux container to mount to Azure Storage. Do not use / or /home .

**Caution**

The directory specified in **Mount path** in the container should be empty. Any content stored in this directory is deleted when the Azure Storage is mounted (if you specify a directory under /home , for example). If you are migrating files for an existing app, make a backup of the app and its content before you begin.

**NOTE**

Adding, editing, or deleting a storage mount causes the app to be restarted.

## Test the mounted storage

To validate that the Azure Storage is mounted successfully for the app:

1. [Open an SSH session](#) into the container.
2. In the SSH terminal, execute the following command:

```
df -h
```

3. Check if the storage share is mounted. If it's not present, there's an issue with mounting the storage share.
4. Check latency or general reachability of the storage mount with the following command:

```
tcpping Storageaccount.file.core.windows.net
```

## Best practices

- To avoid potential issues related to latency, place the app and the Azure Storage account in the same Azure region. Note, however, if the app and Azure Storage account are in same Azure region, and if you grant access from App Service IP addresses in the [Azure Storage firewall configuration](#), then these IP restrictions are not honored.
- The mount directory in the container app should be empty. Any content stored at this path is deleted when the Azure Storage is mounted. If you are migrating files for an existing app, make a backup of the app and its content before you begin.
- The mount directory in the container app should be empty. Any content stored at this path is deleted when the Azure Storage is mounted (if you specify a directory under `/home`, for example). If you are migrating files for an existing app, make a backup of the app and its content before you begin.
- Mounting the storage to `/home` is not recommended because it may result in performance bottlenecks for the app.
- In the Azure Storage account, avoid [regenerating the access key](#) that's used to mount the storage in the app. The storage account contains two different keys. Use a stepwise approach to ensure that the storage mount remains available to the app during key regeneration. For example, assuming that you used **key1** to configure storage mount in your app:
  1. Regenerate **key2**.
  2. In the storage mount configuration, update the access the key to use the regenerated **key2**.
  3. Regenerate **key1**.
- If you delete an Azure Storage account, container, or share, remove the corresponding storage mount configuration in the app to avoid possible error scenarios.
- The mounted Azure Storage account can be either Standard or Premium performance tier. Based on the app capacity and throughput requirements, choose the appropriate performance tier for the storage account. See the scalability and performance targets that correspond to the storage type:
  - [For Files](#) (Windows and Linux containers)
  - [For Blobs](#) (Linux containers only)
- If your app [scales to multiple instances](#), all the instances connect to the same mounted Azure Storage account. To avoid performance bottlenecks and throughput issues, choose the appropriate performance tier for the storage account.
- It's not recommended to use storage mounts for local databases (such as SQLite) or for any other applications and components that rely on file handles and locks.
- When using Azure Storage [private endpoints](#) with the app, you need to set the following two app settings:
  - `WEBSITE_DNS_SERVER = 168.63.129.16`
  - `WEBSITE_VNET_ROUTE_ALL = 1`
- If you [initiate a storage failover](#) and the storage account is mounted to the app, the mount will fail to connect until you either restart the app or remove and add the Azure Storage mount.

## Next steps

- [Migrate custom software to Azure App Service using a custom container](#).
- [Configure a custom container](#).

# Run your app in Azure App Service directly from a ZIP package

11/2/2021 • 4 minutes to read • [Edit Online](#)

In [Azure App Service](#), you can run your apps directly from a deployment ZIP package file. This article shows how to enable this functionality in your app.

All other deployment methods in App Service have something in common: your files are deployed to `D:\home\site\wwwroot` in your app (or `/home/site/wwwroot` for Linux apps). Since the same directory is used by your app at runtime, it's possible for deployment to fail because of file lock conflicts, and for the app to behave unpredictably because some of the files are not yet updated.

In contrast, when you run directly from a package, the files in the package are not copied to the `wwwroot` directory. Instead, the ZIP package itself gets mounted directly as the read-only `wwwroot` directory. There are several benefits to running directly from a package:

- Eliminates file lock conflicts between deployment and runtime.
- Ensures only full-deployed apps are running at any time.
- Can be deployed to a production app (with restart).
- Improves the performance of Azure Resource Manager deployments.
- May reduce cold-start times, particularly for JavaScript functions with large npm package trees.

## NOTE

Currently, only ZIP package files are supported.

## Create a project ZIP package

### NOTE

If you downloaded the files in a ZIP package, extract the files first. For example, if you downloaded a ZIP package from GitHub, you cannot deploy that file as-is. GitHub adds additional nested directories, which do not work with App Service.

In a local terminal window, navigate to the root directory of your app project.

This directory should contain the entry file to your web app, such as `index.html`, `index.php`, and `app.js`. It can also contain package management files like `project.json`, `composer.json`, `package.json`, `bower.json`, and `requirements.txt`.

Unless you want App Service to run deployment automation for you, run all the build tasks (for example, `npm`, `bower`, `gulp`, `composer`, and `pip`) and make sure that you have all the files you need to run the app. This step is required if you want to [run your package directly](#).

Create a ZIP archive of everything in your project. For `dotnet` projects, this folder is the output folder of the `dotnet publish` command. The following command uses the default tool in your terminal:

```
Bash
zip -r <file-name>.zip .

PowerShell
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

## Enable running from package

The `WEBSITE_RUN_FROM_PACKAGE` app setting enables running from a package. To set it, run the following command with Azure CLI.

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings
WEBSITE_RUN_FROM_PACKAGE="1"
```

`WEBSITE_RUN_FROM_PACKAGE="1"` lets you run your app from a package local to your app. You can also [run from a remote package](#).

## Run the package

The easiest way to run a package in your App Service is with the Azure CLI [az webapp deployment source config-zip](#) command. For example:

```
az webapp deployment source config-zip --resource-group <group-name> --name <app-name> --src <filename>.zip
```

Because the `WEBSITE_RUN_FROM_PACKAGE` app setting is set, this command doesn't extract the package content to the `D:\home\site\wwwroot` directory of your app. Instead, it uploads the ZIP file as-is to `D:\home\data\SitePackages`, and creates a `packagename.txt` in the same directory, that contains the name of the ZIP package to load at runtime. If you upload your ZIP package in a different way (such as [FTP](#)), you need to create the `D:\home\data\SitePackages` directory and the `packagename.txt` file manually.

The command also restarts the app. Because `WEBSITE_RUN_FROM_PACKAGE` is set, App Service mounts the uploaded package as the read-only `wwwroot` directory and runs the app directly from that mounted directory.

## Run from external URL instead

You can also run a package from an external URL, such as Azure Blob Storage. You can use the [Azure Storage Explorer](#) to upload package files to your Blob storage account. You should use a private storage container with a [Shared Access Signature \(SAS\)](#) or [use a managed identity](#) to enable the App Service runtime to access the package securely.

Once you upload your file to Blob storage and have an SAS URL for the file, set the `WEBSITE_RUN_FROM_PACKAGE` app setting to the URL. The following example does it by using Azure CLI:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
WEBSITE_RUN_FROM_PACKAGE="https://myblobstorage.blob.core.windows.net/content/SampleCoreMVCApp.zip?st=2018-
02-13T09%3A48%3A00Z&se=2044-06-14T09%3A48%3A00Z&sp=r&sv=2017-04-
17&sr=b&sig=bNrVrEFzRHQB17GFJ7boEanetyJ9DGwBSV8OM3Mdh%2FM%3D"
```

If you publish an updated package with the same name to Blob storage, you need to restart your app so that the updated package is loaded into App Service.

### Fetch a package from Azure Blob Storage using a managed identity

Azure Blob Storage can be configured to [authorize requests with Azure AD](#). This means that instead of

generating a SAS key with an expiration, you can instead rely on the application's [managed identity](#). By default, the app's system-assigned identity will be used. If you wish to specify a user-assigned identity, you can set the `WEBSITE_RUN_FROM_PACKAGE_BLOB_MI_RESOURCE_ID` app setting to the resource ID of that identity. The setting can also accept "SystemAssigned" as a value, although this is the same as omitting the setting altogether.

To enable the package to be fetched using the identity:

1. Ensure that the blob is [configured for private access](#).
2. Grant the identity the [Storage Blob Data Reader](#) role with scope over the package blob. See [Assign an Azure role for access to blob data](#) for details on creating the role assignment.
3. Set the `WEBSITE_RUN_FROM_PACKAGE` application setting to the blob URL of the package. This will likely be of the form "`https://<storage-account-name>.blob.core.windows.net/<container-name>/<path-to-package>`" or similar.

## Troubleshooting

- Running directly from a package makes `wwwroot` read-only. Your app will receive an error if it tries to write files to this directory.
- TAR and GZIP formats are not supported.
- The ZIP file can be at most 1GB
- This feature is not compatible with [local cache](#).
- For improved cold-start performance, use the local Zip option (`WEBSITE_RUN_FROM_PACKAGE =1`).

## More resources

- [Continuous deployment for Azure App Service](#)
- [Deploy code with a ZIP or WAR file](#)

# Deploy files to App Service

11/2/2021 • 10 minutes to read • [Edit Online](#)

This article shows you how to deploy your code as a ZIP, WAR, JAR, or EAR package to [Azure App Service](#). It also shows how to deploy individual files to App Service, separate from your application package.

## Prerequisites

To complete the steps in this article, [create an App Service app](#), or use an app that you created for another tutorial.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create a project ZIP package

### NOTE

If you downloaded the files in a ZIP package, extract the files first. For example, if you downloaded a ZIP package from GitHub, you cannot deploy that file as-is. GitHub adds additional nested directories, which do not work with App Service.

In a local terminal window, navigate to the root directory of your app project.

This directory should contain the entry file to your web app, such as *index.html*, *index.php*, and *app.js*. It can also contain package management files like *project.json*, *composer.json*, *package.json*, *bower.json*, and *requirements.txt*.

Unless you want App Service to run deployment automation for you, run all the build tasks (for example, `npm`, `bower`, `gulp`, `composer`, and `pip`) and make sure that you have all the files you need to run the app. This step is required if you want to [run your package directly](#).

Create a ZIP archive of everything in your project. For `dotnet` projects, this folder is the output folder of the `dotnet publish` command. The following command uses the default tool in your terminal:

```
Bash
zip -r <file-name>.zip .

PowerShell
Compress-Archive -Path * -DestinationPath <file-name>.zip
```

## Deploy a ZIP package

When you deploy a ZIP package, App Service unpacks its contents in the default path for your app (`D:\home\site\wwwroot` for Windows, `/home/site/wwwroot` for Linux).

This ZIP package deployment uses the same Kudu service that powers continuous integration-based deployments. Kudu supports the following functionality for ZIP package deployment:

- Deletion of files left over from a previous deployment.
- Option to turn on the default build process, which includes package restore.
- Deployment customization, including running deployment scripts.
- Deployment logs.

- A package size limit of 2048 MB.

For more information, see [Kudu documentation](#).

#### NOTE

Files in the ZIP package are copied only if their timestamps don't match what is already deployed. Generating a zip using a build process that caches outputs can result in faster deployments. See [Deploying from a zip file or url](#), for more information.

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Kudu API](#)
- [Kudu UI](#)

Deploy a ZIP package to your web app by using the `az webapp deploy` command. The CLI command uses the [Kudu publish API](#) to deploy the files and can be fully customized.

The following example pushes a ZIP package to your site. Specify the path to your local ZIP package for `--src-path`.

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path <zip-package-path>
```

This command restarts the app after deploying the ZIP package.

Depending on your web apps's networking configuration, direct access to the site from your local environment may be blocked. To deploy your code in this scenario, you can publish your ZIP to a storage system accessible from the web app and trigger the app to *pull* the ZIP from the storage location, instead of *pushing* the ZIP to the web app. See [this article on deploying to network secured web apps](#) for more information.

The following example uses the `--src-url` parameter to specify the URL of an Azure Storage account that the site should pull the ZIP from.

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-url
"https://storagesample.blob.core.windows.net/sample-container/myapp.zip?sv=2021-10-
01&sb&sig=slk22f3UrS823n4kSh8Skjpa7Naj4CG3"
```

## Enable build automation for ZIP deploy

By default, the deployment engine assumes that a ZIP package is ready to run as-is and doesn't run any build automation. To enable the same build automation as in a [Git deployment](#), set the

`SCM_DO_BUILD_DURING_DEPLOYMENT` app setting by running the following command in the [Cloud Shell](#):

```
az webapp config appsettings set --resource-group <group-name> --name <app-name> --settings
SCM_DO_BUILD_DURING_DEPLOYMENT=true
```

For more information, see [Kudu documentation](#).

## What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. So the deployment can fail because of locked files. The app might also behave unpredictably during deployment because the files aren't all updated at the same time. This

behavior is undesirable for a customer-facing app. There are a few ways to avoid these issues:

- [Run your app directly from the ZIP package](#), without unpacking it.
- Stop your app or enable offline mode for it during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) turned on.

## Deploy WAR/JAR/EAR packages

You can deploy your [WAR](#), [JAR](#), or [EAR](#) package to App Service to run your Java web app using the Azure CLI, PowerShell, or the Kudu publish API.

The deployment process places the package on the shared file drive correctly (see [Kudu publish API reference](#)). For that reason, deploying WAR/JAR/EAR packages using [FTP](#) or WebDeploy is not recommended.

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Kudu API](#)
- [Kudu UI](#)

Deploy a WAR package to Tomcat or JBoss EAP by using the [az webapp deploy](#) command. Specify the path to your local Java package for `--src-path`.

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-path ./<package-name>.war
```

Depending on your web apps's networking configuration, direct access to the site from your local environment may be blocked. To deploy your code in this scenario, you can publish your ZIP to a storage system accessible from the web app and trigger the app to *pull* the ZIP from the storage location, instead of *pushing* the ZIP to the web app. See [this article on deploying to network secured web apps](#) for more information.

The following example uses the `--src-url` parameter to specify the URL of an Azure Storage account that the web app should pull the ZIP from.

```
az webapp deploy --resource-group <group-name> --name <app-name> --src-url
"https://storagesample.blob.core.windows.net/sample-container/myapp.war?sv=2021-10-
01&sb&sig=slk22f3UrS823n4kSh8Skjpa7Naj4CG3
```

The CLI command uses the [Kudu publish API](#) to deploy the package and can be fully customized.

## Deploy individual files

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Kudu API](#)
- [Kudu UI](#)

Deploy a startup script, library, and static file to your web app by using the [az webapp deploy](#) command with the `--type` parameter.

If you deploy a startup script this way, App Service automatically uses your script to start your app.

The CLI command uses the [Kudu publish API](#) to deploy the files and can be fully customized.

### Deploy a startup script

```
az webapp deploy --resource group <group-name> --name <app-name> --src-path scripts/startup.sh --type=startup
```

## Deploy a library file

```
az webapp deploy --resource group <group-name> --name <app-name> --src-path driver.jar --type=lib
```

## Deploy a static file

```
az webapp deploy --resource group <group-name> --name <app-name> --src-path config.json --type=static
```

# Kudu publish API reference

The `publish` Kudu API allows you to specify the same parameters from the CLI command as URL query parameters. To authenticate with the Kudu API, you can use basic authentication with your app's [deployment credentials](#).

The table below shows the available query parameters, their allowed values, and descriptions.

KEY	ALLOWED VALUES	DESCRIPTION	REQUIRED	TYPE
<code>type</code>	<code>war</code>   <code>jar</code>   <code>ear</code>   <code>lib</code>   <code>startup</code>   <code>static</code>   <code>zip</code>	<p>The type of the artifact being deployed, this sets the default target path and informs the web app how the deployment should be handled.</p> <ul style="list-style-type: none"><li>- <code>type=zip</code> : Deploy a ZIP package by unzipping the content to <code>/home/site/wwwroot</code>. The <code>path</code> parameter is optional.</li><li>- <code>type=war</code> : Deploy a WAR package. By default, the WAR package is deployed to <code>/home/site/wwwroot/app.war</code>. The target path can be specified with <code>path</code>.</li><li>- <code>type=jar</code> : Deploy a JAR package to <code>/home/site/wwwroot/app.jar</code>. The <code>path</code> parameter is ignored.</li><li>- <code>type=ear</code> : Deploy an EAR package to <code>/home/site/wwwroot/app.ear</code>. The <code>path</code> parameter is ignored.</li><li>- <code>type=lib</code> : Deploy a JAR library file. By default the file is</li></ul>	Yes	String

KEY	ALLOWED VALUES	DESCRIPTION	REQUIRED	TYPE
		<p>By default, the file is deployed to <code>/home/site/libs</code>.</p> <p>The target path can be specified with <code>path</code>.</p> <ul style="list-style-type: none"> <li>- <code>type=static</code> : Deploy a static file (e.g. a script). By default, the file is deployed to <code>/home/site/scripts</code>. The target path can be specified with <code>path</code>.</li> <li>- <code>type=startup</code> : Deploy a script that App Service automatically uses as the startup script for your app. By default, the script is deployed to <code>D:\home\site\scripts\&lt;name-of-source&gt;</code> for Windows and <code>home/site/wwwroot/startup.sh</code> for Linux. The target path can be specified with <code>path</code>.</li> </ul>		
<code>restart</code>	<code>true</code>   <code>false</code>	<p>By default, the API restarts the app following the deployment operation (<code>restart=true</code>). To deploy multiple artifacts, prevent restarts on the all but the final deployment by setting <code>restart=false</code>.</p>	No	Boolean
<code>clean</code>	<code>true</code>   <code>false</code>	Specifies whether to clean (delete) the target deployment before deploying the artifact there.	No	Boolean

KEY	ALLOWED VALUES	DESCRIPTION	REQUIRED	TYPE
<code>ignorestack</code>	<code>true   false</code>	The publish API uses the <code>WEBSITE_STACK</code> environment variable to choose safe defaults depending on your site's language stack. Setting this parameter to <code>false</code> disables any language-specific defaults.	No	Boolean
<code>path</code>	<code>"&lt;absolute-path&gt;"</code>	The absolute path to deploy the artifact to. For example, <code>"/home/site/deployments/tools driver.jar"</code> <code>,</code> <code>"/home/site/scripts/helper.sh"</code> <code>.</code>	No	String

## Next steps

For more advanced deployment scenarios, try [deploying to Azure with Git](#). Git-based deployment to Azure enables version control, package restore, MSBuild, and more.

## More resources

- [Kudu: Deploying from a zip file](#)
- [Azure App Service Deployment Credentials](#)
- [Environment variables and app settings reference](#)

# Deploy your app to Azure App Service using FTP/S

11/2/2021 • 5 minutes to read • [Edit Online](#)

This article shows you how to use FTP or FTPS to deploy your web app, mobile app backend, or API app to [Azure App Service](#).

The FTP/S endpoint for your app is already active. No configuration is necessary to enable FTP/S deployment.

## NOTE

The **Development Center (Classic)** page in the Azure portal, which is the old deployment experience, will be deprecated in March, 2021. This change will not affect any existing deployment settings in your app, and you can continue to manage app deployment in the **Deployment Center** page.

## Get deployment credentials

- Follow the instructions at [Configure deployment credentials for Azure App Service](#) to copy the application-scope credentials or set the user-scope credentials. You can connect to the FTP/S endpoint of your app using either credentials.
- Craft the FTP username in the following format, depending on your choice of credential scope:

APPLICATION-SCOPE	USER-SCOPE
<app-name>\\$<app-name>	<app-name>\<deployment-user>

In App Service, the FTP/S endpoint is shared among apps. Because the user-scope credentials aren't linked to a specific resource, you need to prepend the user-scope username with the app name as shown above.

## Get FTP/S endpoint

- [Azure portal](#)
- [Azure CLI](#)
- [Azure PowerShell](#)

In the same management page for your app where you copied the deployment credentials (**Deployment Center > FTP Credentials**), copy the **FTPS endpoint**.

## Deploy files to Azure

- From your FTP client (for example, [Visual Studio](#), [Cyberduck](#), or [WinSCP](#)), use the connection information you gathered to connect to your app.
- Copy your files and their respective directory structure to the [/site/wwwroot directory](#) in Azure (or the [/site/wwwroot/App\\_Data/Jobs/](#) directory for WebJobs).
- Browse to your app's URL to verify the app is running properly.

## NOTE

Unlike [Git-based deployments](#) and [Zip deployment](#), FTP deployment doesn't support build automation, such as:

- dependency restores (such as NuGet, NPM, PIP and Composer automations)
- compilation of .NET binaries
- generation of web.config (here is a [Node.js example](#))

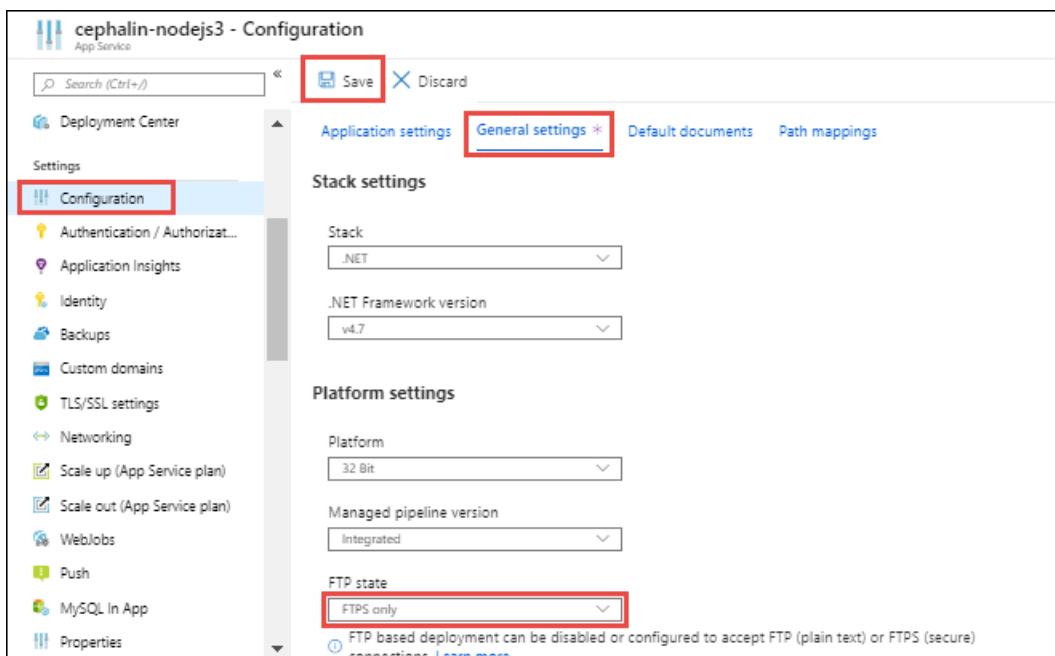
Generate these necessary files manually on your local machine, and then deploy them together with your app.

## Enforce FTPS

For enhanced security, you should allow FTP over TLS/SSL only. You can also disable both FTP and FTPS if you don't use FTP deployment.

- [Azure portal](#)
- [Azure CLI](#)
- [Azure PowerShell](#)

1. In your app's resource page in [Azure portal](#), select **Configuration > General settings** from the left navigation.
2. To disable unencrypted FTP, select **FTPS Only** in **FTP state**. To disable both FTP and FTPS entirely, select **Disabled**. When finished, click **Save**. If using **FTPS Only**, you must enforce TLS 1.2 or higher by navigating to the **TLS/SSL settings** blade of your web app. TLS 1.0 and 1.1 are not supported with **FTPS Only**.



## What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. So the deployment can fail because of locked files. The app might also behave unpredictably during deployment because the files aren't all updated at the same time. This behavior is undesirable for a customer-facing app. There are a few ways to avoid these issues:

- [Run your app directly from the ZIP package](#), without unpacking it.
- Stop your app or enable offline mode for it during deployment. For more information, see [Deal with locked](#)

files during deployment.

- Deploy to a [staging slot](#) with [auto swap](#) turned on.

## Troubleshoot FTP deployment

- [How can I troubleshoot FTP deployment?](#)
- [I'm not able to FTP and publish my code. How can I resolve the issue?](#)
- [How can I connect to FTP in Azure App Service via passive mode?](#)

### **How can I troubleshoot FTP deployment?**

The first step for troubleshooting FTP deployment is isolating a deployment issue from a runtime application issue.

A deployment issue typically results in no files or wrong files deployed to your app. You can troubleshoot by investigating your FTP deployment or selecting an alternate deployment path (such as source control).

A runtime application issue typically results in the right set of files deployed to your app but incorrect app behavior. You can troubleshoot by focusing on code behavior at runtime and investigating specific failure paths.

To determine a deployment or runtime issue, see [Deployment vs. runtime issues](#).

### **I'm not able to FTP and publish my code. How can I resolve the issue?**

Check that you've entered the correct [hostname](#) and [credentials](#). Check also that the following FTP ports on your machine are not blocked by a firewall:

- FTP control connection port: 21, 990
- FTP data connection port: 989, 10001-10300

### **How can I connect to FTP in Azure App Service via passive mode?**

Azure App Service supports connecting via both Active and Passive mode. Passive mode is preferred because your deployment machines are usually behind a firewall (in the operating system or as part of a home or business network). See an [example from the WinSCP documentation](#).

## More resources

- [Local Git deployment to Azure App Service](#)
- [Azure App Service Deployment Credentials](#)
- [Sample: Create a web app and deploy files with FTP \(Azure CLI\)](#).
- [Sample: Upload files to a web app using FTP \(PowerShell\)](#).

# Sync content from a cloud folder to Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to sync your content to [Azure App Service](#) from Dropbox and OneDrive.

With the content sync approach, your work with your app code and content in a designated cloud folder to make sure it's in a ready-to-deploy state, and then sync to App Service with the click of a button.

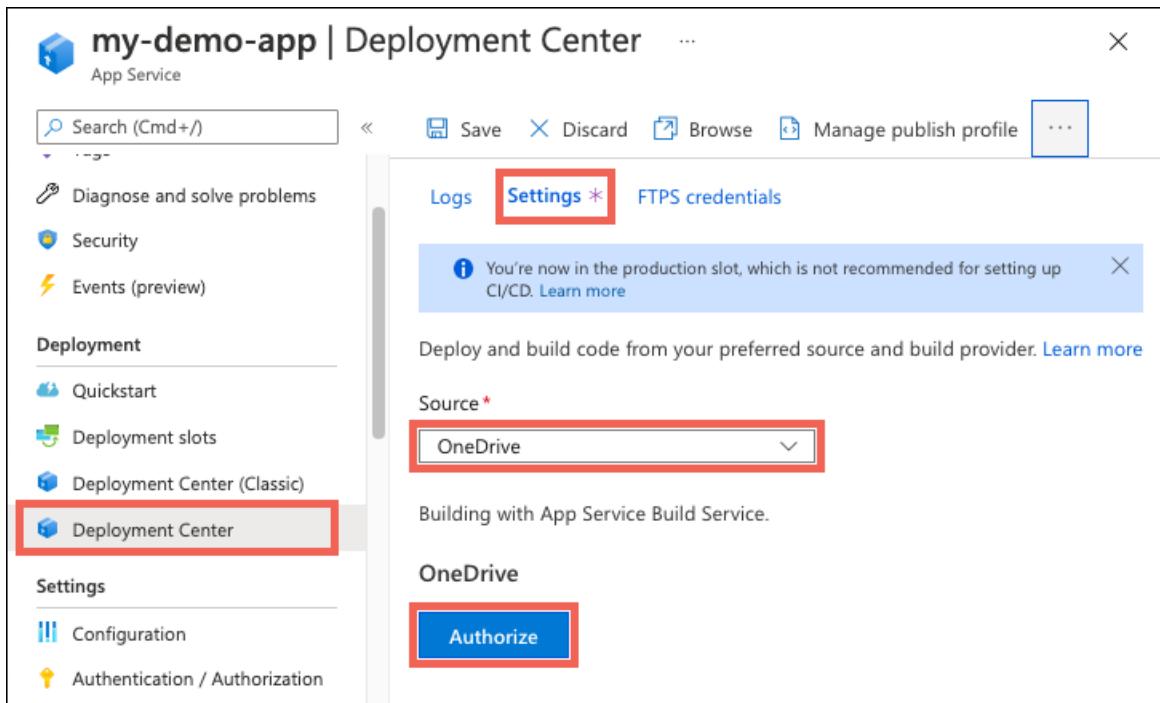
Because of underlying differences in the APIs, [OneDrive for Business](#) is not supported at this time.

## NOTE

The [Development Center \(Classic\)](#) page in the Azure portal, which is the old deployment experience, will be deprecated in March, 2021. This change will not affect any existing deployment settings in your app, and you can continue to manage app deployment in the [Deployment Center](#) page.

## Enable content sync deployment

1. In the [Azure portal](#), navigate to the management page for your App Service app.
2. From the left menu, click **Deployment Center** > **Settings**.
3. In **Source**, select **OneDrive or Dropbox**.
4. Click **Authorize** and follow the authorization prompts.



You only need to authorize with OneDrive or Dropbox once for your Azure account. To authorize a different OneDrive or Dropbox account for an app, click **Change account**.

5. In **Folder**, select the folder to synchronize. This folder is created under the following designated content path in OneDrive or Dropbox.

- OneDrive: Apps\Azure Web Apps

- Dropbox: Apps\Azure

6. Click Save.

## Synchronize content

- Azure portal
- Azure CLI
- Azure PowerShell

1. In the [Azure portal](#), navigate to the management page for your App Service app.

2. From the left menu, click Deployment Center > Redeploy/Sync.

The screenshot shows the Azure portal's Deployment Center interface for an app named 'my-demo-app'. The left sidebar has 'Deployment Center' selected. The top navigation bar includes 'Redeploy/Sync' which is highlighted with a red box. The main content area displays deployment logs for a specific date, with the 'Logs' tab selected. A log entry from 'Friday, February 26, 2021' is shown, indicating a successful sync from OneDrive.

3. Click OK to confirm the sync.

## Disable content sync deployment

1. In the [Azure portal](#), navigate to the management page for your App Service app.

2. From the left menu, click Deployment Center > Settings > Disconnect.

The screenshot shows the Azure portal's Deployment Center interface for an app named 'my-demo-app'. The left sidebar has 'Deployment Center' selected. The top navigation bar has 'Settings' highlighted with a red box. The main content area shows deployment settings, with the 'Source' section set to 'OneDrive' and the 'Disconnect' button highlighted with a red box.

## What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. So the deployment can fail because of locked files. The app might also behave unpredictably during deployment because the files aren't all updated at the same time. This behavior is undesirable for a customer-facing app. There are a few ways to avoid these issues:

- Run your app directly from the ZIP package, without unpacking it.
- Stop your app or enable offline mode for it during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a staging slot with [auto swap](#) turned on.

## Next steps

[Deploy from local Git repo](#)

# Continuous deployment to Azure App Service

11/2/2021 • 6 minutes to read • [Edit Online](#)

Azure App Service enables continuous deployment from [GitHub](#), [Bitbucket](#), and [Azure Repos](#) repositories by pulling in the latest updates.

## NOTE

The [Development Center \(Classic\)](#) page in the Azure portal, an earlier version of the deployment functionality, was deprecated in March 2021. This change doesn't affect existing deployment settings in your app, and you can continue to manage app deployment from the [Deployment Center](#) page in the portal.

## Prepare your repository

To get automated builds from Azure App Service build server, make sure that your repository root has the correct files in your project.

RUNTIME	ROOT DIRECTORY FILES
ASP.NET (Windows only)	<code>*.sln</code> , <code>*.csproj</code> , or <code>default.aspx</code>
ASP.NET Core	<code>*.sln</code> or <code>*.csproj</code>
PHP	<code>index.php</code>
Ruby (Linux only)	<code>Gemfile</code>
Node.js	<code>server.js</code> , <code>app.js</code> , or <code>package.json</code> with a start script
Python	<code>*.py</code> , <code>requirements.txt</code> , or <code>runtime.txt</code>
HTML	<code>default.htm</code> , <code>default.html</code> , <code>default.asp</code> , <code>index.htm</code> , <code>index.html</code> , or <code>iisstart.htm</code>
WebJobs	<code>&lt;job_name&gt;/run.&lt;extension&gt;</code> under <code>App_Data/jobs/continuous</code> for continuous WebJobs, or <code>App_Data/jobs/triggered</code> for triggered WebJobs. For more information, see <a href="#">Kudu WebJobs documentation</a> .
Functions	See <a href="#">Continuous deployment for Azure Functions</a> .

To customize your deployment, include a `.deployment` file in the repository root. For more information, see [Customize deployments](#) and [Custom deployment script](#).

## NOTE

If you use Visual Studio, let [Visual Studio create a repository for you](#). Your project will immediately be ready for deployment via Git.

# Configure the deployment source

1. In the [Azure portal](#), go to the management page for your App Service app.
2. In the left pane, select **Deployment Center**. Then select **Settings**.
3. In the **Source** box, select one of the CI/CD options:

The screenshot shows the 'my-demo-app | Deployment Center' page in the Azure portal. The 'Settings' tab is selected. On the left, there's a sidebar with 'Deployment Center' highlighted. Below it is a 'Source' dropdown menu with several options listed:

- Continuous Deployment (CI/CD)
  - GitHub
  - Bitbucket
  - Local Git
  - Azure Repos
- Manual Deployment (Push)
  - External Git
  - OneDrive
  - Dropbox

Select the tab that corresponds to your build provider to continue.

- [GitHub](#)
- [Bitbucket](#)
- [Local Git](#)
- [Azure Repos](#)

4. [GitHub Actions](#) is the default build provider. To change the provider, select **Change provider** > **App Service Build Service (Kudu)** > **OK**.

#### NOTE

To use Azure Pipelines as the build provider for your App Service app, configure CI/CD directly from Azure Pipelines. Don't configure it in App Service. The [Azure Pipelines](#) option just points you in the right direction.

5. If you're deploying from GitHub for the first time, select **Authorize** and follow the authorization prompts. If you want to deploy from a different user's repository, select **Change Account**.
6. After you authorize your Azure account with GitHub, select the **Organization**, **Repository**, and **Branch** to configure CI/CD for. If you can't find an organization or repository, you might need to enable more permissions on GitHub. For more information, see [Managing access to your organization's repositories](#).
7. When GitHub Actions is selected as the build provider, you can select the workflow file you want by using the **Runtime stack** and **Version** dropdown lists. Azure commits this workflow file into your selected GitHub repository to handle build and deploy tasks. To see the file before saving your changes, select **Preview file**.

#### NOTE

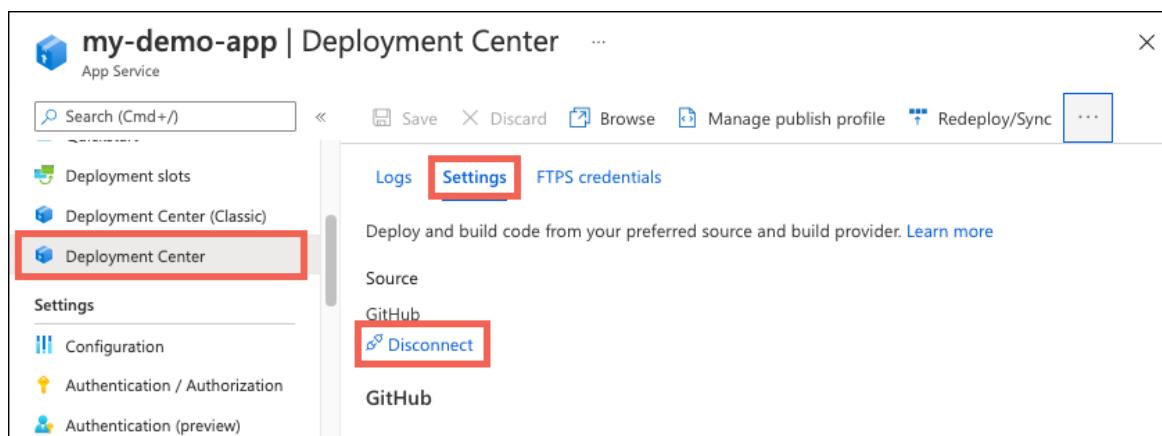
App Service detects the [language stack setting](#) of your app and selects the most appropriate workflow template. If you choose a different template, it might deploy an app that doesn't run properly. For more information, see [How the GitHub Actions build provider works](#).

#### 8. Select Save.

New commits in the selected repository and branch now deploy continuously into your App Service app. You can track the commits and deployments on the [Logs](#) tab.

## Disable continuous deployment

1. In the [Azure portal](#), go to the management page for your App Service app.
2. In the left pane, select **Deployment Center**. Then select **Settings > Disconnect**:



3. By default, the GitHub Actions workflow file is preserved in your repository, but it will continue to trigger deployment to your app. To delete the file from your repository, select **Delete workflow file**.
4. Select **OK**.

## What happens to my app during deployment?

All the officially supported deployment methods make changes to the files in the `/home/site/wwwroot` folder of your app. These files are used to run your app. So the deployment can fail because of locked files. The app might also behave unpredictably during deployment because the files aren't all updated at the same time. This behavior is undesirable for a customer-facing app. There are a few ways to avoid these issues:

- [Run your app directly from the ZIP package](#), without unpacking it.
- Stop your app or enable offline mode for it during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) turned on.

## How the GitHub Actions build provider works

The GitHub Actions build provider is an option for [CI/CD from GitHub](#). It completes these actions to set up CI/CD:

- Deposits a GitHub Actions workflow file into your GitHub repository to handle build and deploy tasks to App Service.
- Adds the publishing profile for your app as a GitHub secret. The workflow file uses this secret to authenticate with App Service.

- Captures information from the [workflow run logs](#) and displays it on the **Logs** tab in your app's Deployment Center.

You can customize the GitHub Actions build provider in these ways:

- Customize the workflow file after it's generated in your GitHub repository. For more information, see [Workflow syntax for GitHub Actions](#). Just make sure that the workflow deploys to App Service with the `azure/webapps-deploy` action.
- If the selected branch is protected, you can still preview the workflow file without saving the configuration and then manually add it into your repository. This method doesn't give you log integration with the Azure portal.
- Instead of using a publishing profile, deploy by using a [service principal](#) in Azure Active Directory.

#### **Authenticate by using a service principal**

This optional configuration replaces the default authentication with publishing profiles in the generated workflow file.

1. Generate a service principal by using the `az ad sp create-for-rbac` command in the [Azure CLI](#). In the following example, replace <subscription-id>, <group-name>, and <app-name> with your own values:

```
az ad sp create-for-rbac --name "myAppDeployAuth" --role contributor \
 --scopes /subscriptions/<subscription-id>/resourceGroups/<group-name>/providers/Microsoft.Web/sites/<app-name> \
 --sdk-auth
```

#### **IMPORTANT**

For security, grant the minimum required access to the service principal. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

2. Save the entire JSON output for the next step, including the top-level `{}`.
3. In [GitHub](#), in your repository, select **Settings > Secrets > Add a new secret**.
4. Paste the entire JSON output from the Azure CLI command into the secret's value field. Give the secret a name like `AZURE_CREDENTIALS`.
5. In the workflow file generated by the Deployment Center, revise the `azure/webapps-deploy` step to look like the following example (which is modified from a Node.js workflow file):

```
- name: Sign in to Azure
Use the GitHub secret you added.
- uses: azure/login@v1
 with:
 creds: ${{ secrets.AZURE_CREDENTIALS }}
- name: Deploy to Azure Web App
Remove publish-profile.
- uses: azure/webapps-deploy@v2
 with:
 app-name: '<app-name>'
 slot-name: 'production'
 package: .
- name: Sign out of Azure.
 run:
 az logout
```

## Deploy from other repositories

For Windows apps, you can manually configure continuous deployment from a cloud Git or Mercurial repository that the portal doesn't directly support, like [GitLab](#). You do that by selecting **External Git** in the **Source** dropdown list. For more information, see [Set up continuous deployment using manual steps](#).

## More resources

- [Deploy from Azure Pipelines to Azure App Services](#)
- [Investigate common problems with continuous deployment](#)
- [Use Azure PowerShell](#)
- [Project Kudu](#)

# Continuous deployment with custom containers in Azure App Service

11/2/2021 • 9 minutes to read • [Edit Online](#)

In this tutorial, you configure continuous deployment for a custom container image from managed [Azure Container Registry](#) repositories or [Docker Hub](#).

## 1. Go to Deployment Center

In the [Azure portal](#), navigate to the management page for your App Service app.

From the left menu, click **Deployment Center > Settings**.

## 2. Choose deployment source

Choose the deployment source depends on your scenario:

- **Container registry** sets up CI/CD between your container registry and App Service.
- The **GitHub Actions** option is for you if you maintain the source code for your container image in GitHub. Triggered by new commits to your GitHub repository, the deploy action can run `docker build` and `docker push` directly to your container registry, then update your App Service app to run the new image. For more information, see [How CI/CD works with GitHub Actions](#).
- To set up CI/CD with **Azure Pipelines**, see [Deploy an Azure Web App Container from Azure Pipelines](#).

### NOTE

For a Docker Compose app, select **Container Registry**.

If you choose GitHub Actions, **click Authorize** and follow the authorization prompts. If you've already authorized with GitHub before, you can deploy from a different user's repository by clicking **Change Account**.

Once you authorize your Azure account with GitHub, **select the Organization, Repository, and Branch** to deploy from.

## 2. Configure registry settings

## 3. Configure registry settings

To deploy a multi-container (Docker Compose) app, **select Docker Compose** in **Container Type**.

If you don't see the **Container Type** dropdown, scroll back up to **Source** and **select Container Registry**.

In **Registry source**, **select** where your container registry is. If it's neither Azure Container Registry nor Docker Hub, **select Private Registry**.

#### NOTE

If your multi-container (Docker Compose) app uses more than one private image, make sure the private images are in the same private registry and accessible with the same user credentials. If your multi-container app only uses public images, **select Docker Hub**, even if some images are not in Docker Hub.

Follow the next steps by selecting the tab that matches your choice.

- [Azure Container Registry](#)
- [Docker Hub](#)
- [Private Registry](#)

The **Registry** dropdown displays the registries in the same subscription as your app. **Select** the registry you want.

#### NOTE

- If want to use Managed Identities to lock down ACR access follow this guide:
  - [How to use system-assigned Managed Identities with App Service and Azure Container Registry](#)
  - [How to use user-assigned Managed Identities with App Service and Azure Container Registry](#)
- To deploy from a registry in a different subscription, **select Private Registry** in **Registry source** instead.

Select the **Image** and **Tag** to deploy. If you want, **type** the start up command in **Startup File**.

Follow the next step depending on the **Container Type**:

- For **Docker Compose**, **select** the registry for your private images. **Click Choose file** to upload your [Docker Compose file](#), or just **paste** the content of your Docker Compose file into **Config**.
- For **Single Container**, **select** the **Image** and **Tag** to deploy. If you want, **type** the start up command in **Startup File**.

App Service appends the string in **Startup File** to **the end of the docker run command** (as the **[COMMAND] [ARG...]** segment) when starting your container.

## 3. Enable CI/CD

## 4. Enable CI/CD

App Service supports CI/CD integration with Azure Container Registry and Docker Hub. To enable it, **select On** in **Continuous deployment**.

#### NOTE

If you select **GitHub Actions** in **Source**, you don't get this option because CI/CD is handled by GitHub Actions directly. Instead, you see a **Workflow Configuration** section, where you can **click Preview file** to inspect the workflow file. Azure commits this file into your selected GitHub source repository to handle build and deploy tasks. For more information, see [How CI/CD works with GitHub Actions](#).

When you enable this option, App Service adds a webhook to your repository in Azure Container Registry or Docker Hub. Your repository posts to this webhook whenever your selected image is updated with **docker push**. The webhook causes your App Service app to restart and run **docker pull** to get the updated image.

For other **private registries**, you can post to the webhook manually or as a step in a CI/CD pipeline. In

Webhook URL, click the **Copy** button to get the webhook URL.

#### NOTE

Support for multi-container (Docker Compose) apps is limited:

- For Azure Container Registry, App Service creates a webhook in the selected registry with the registry as the scope. A `docker push` to any repository in the registry (including the ones not referenced by your Docker Compose file) triggers an app restart. You may want to [modify the webhook](#) to a narrower scope.
- Docker Hub doesn't support webhooks at the registry level. You must **add** the webhooks manually to the images specified in your Docker Compose file.

## 4. Save your settings

## 5. Save your settings

Click **Save**.

## How CI/CD works with GitHub Actions

If you choose **GitHub Actions in Source** (see [Choose deployment source](#)), App Service sets up CI/CD in the following ways:

- Deposits a GitHub Actions workflow file into your GitHub repository to handle build and deploy tasks to App Service.
- Adds the credentials for your private registry as GitHub secrets. The generated workflow file runs the [Azure/docker-login](#) action to sign in with your private registry, then runs `docker push` to deploy to it.
- Adds the publishing profile for your app as a GitHub secret. The generated workflow file uses this secret to authenticate with App Service, then runs the [Azure/webapps-deploy](#) action to configure the updated image, which triggers an app restart to pull in the updated image.
- Captures information from the [workflow run logs](#) and displays it in the **Logs** tab in your app's **Deployment Center**.

You can customize the GitHub Actions build provider in the following ways:

- Customize the workflow file after it's generated in your GitHub repository. For more information, see [Workflow syntax for GitHub Actions](#). Just make sure that the workflow ends with the [Azure/webapps-deploy](#) action to trigger an app restart.
- If the selected branch is protected, you can still preview the workflow file without saving the configuration, then add it and the required GitHub secrets into your repository manually. This method doesn't give you the log integration with the Azure portal.
- Instead of a publishing profile, deploy using a [service principal](#) in Azure Active Directory.

#### Authenticate with a service principal

This optional configuration replaces the default authentication with publishing profiles in the generated workflow file.

Generate a service principal with the `az ad sp create-for-rbac` command in the [Azure CLI](#). In the following example, replace `<subscription-id>`, `<group-name>`, and `<app-name>` with your own values. **Save** the entire JSON output for the next step, including the top-level `{}`.

```
az ad sp create-for-rbac --name "myAppDeployAuth" --role contributor \
 --scopes /subscriptions/<subscription-id>/resourceGroups/<group-name>/providers/Microsoft.Web/sites/<app-name> \
 --sdk-auth
```

### IMPORTANT

For security, grant the minimum required access to the service principal. The scope in the previous example is limited to the specific App Service app and not the entire resource group.

In [GitHub](#), **browse** to your repository, then **select Settings > Secrets > Add a new secret**. Paste the entire JSON output from the Azure CLI command into the secret's value field. **Give** the secret a name like

```
AZURE_CREDENTIALS .
```

In the workflow file generated by the **Deployment Center**, **revise** the `azure/webapps-deploy` step with code like the following example:

```
- name: Sign in to Azure
Use the GitHub secret you added
- uses: azure/login@v1
 with:
 creds: ${{ secrets.AZURE_CREDENTIALS }}
- name: Deploy to Azure Web App
Remove publish-profile
- uses: azure/webapps-deploy@v2
 with:
 app-name: '<app-name>'
 slot-name: 'production'
 images: '<registry-server>/${{ secrets.AzureAppService_ContainerUsername_... }}<image>:${{ github.sha }}'
 - name: Sign out of Azure
 run: |
 az logout
```

## Automate with CLI

To configure the container registry and the Docker image, **run** `az webapp config container set`.

- [Azure Container Registry](#)
- [Docker Hub](#)
- [Private Registry](#)

```
az webapp config container set --name <app-name> --resource-group <group-name> --docker-custom-image-name
'<image>:<tag>' --docker-registry-server-url 'https://<registry-name>.azurecr.io' --docker-registry-server-user
'<username>' --docker-registry-server-password '<password>'
```

To configure a multi-container (Docker Compose) app, **prepare** a Docker Compose file locally, then **run** `az webapp config container set` with the `--multicontainer-config-file` parameter. If your Docker Compose file contains private images, **add** `--docker-registry-server-*` parameters as shown in the previous example.

```
az webapp config container set --resource-group <group-name> --name <app-name> --multicontainer-config-file
<docker-compose-file>
```

To configure CI/CD from the container registry to your app, **run** `az webapp deployment container config` with

the `--enable-cd` parameter. The command outputs the webhook URL, but you must create the webhook in your registry manually in a separate step. The following example enables CI/CD on your app, then uses the webhook URL in the output to create the webhook in Azure Container Registry.

```
ci_cd_url=$(az webapp deployment container config --name <app-name> --resource-group <group-name> --enable-cd true --query CI_CD_URL --output tsv)

az acr webhook create --name <webhook-name> --registry <registry-name> --resource-group <group-name> --actions push --uri $ci_cd_url --scope '<image>:<tag>'
```

## More resources

- [Azure Container Registry](#)
- [Create a .NET Core web app in App Service on Linux](#)
- [Quickstart: Run a custom container on App Service](#)
- [App Service on Linux FAQ](#)
- [Configure custom containers](#)
- [Actions workflows to deploy to Azure](#)

# Local Git deployment to Azure App Service

11/2/2021 • 5 minutes to read • [Edit Online](#)

This how-to guide shows you how to deploy your app to [Azure App Service](#) from a Git repository on your local computer.

## Prerequisites

To follow the steps in this how-to guide:

- If you don't have an [Azure subscription](#), create a [free account](#) before you begin.
- [Install Git](#).
- Have a local Git repository with code you want to deploy. To download a sample repository, run the following command in your local terminal window:

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

## Prepare your repository

To get automated builds from Azure App Service build server, make sure that your repository root has the correct files in your project.

RUNTIME	ROOT DIRECTORY FILES
ASP.NET (Windows only)	<code>*.sln</code> , <code>*.csproj</code> , or <code>default.aspx</code>
ASP.NET Core	<code>*.sln</code> or <code>*.csproj</code>
PHP	<code>index.php</code>
Ruby (Linux only)	<code>Gemfile</code>
Node.js	<code>server.js</code> , <code>app.js</code> , or <code>package.json</code> with a start script
Python	<code>*.py</code> , <code>requirements.txt</code> , or <code>runtime.txt</code>
HTML	<code>default.htm</code> , <code>default.html</code> , <code>default.asp</code> , <code>index.htm</code> , <code>index.html</code> , or <code>iisstart.htm</code>
WebJobs	<code>&lt;job_name&gt;/run.&lt;extension&gt;</code> under <code>App_Data/jobs/continuous</code> for continuous WebJobs, or <code>App_Data/jobs/triggered</code> for triggered WebJobs. For more information, see <a href="#">Kudu WebJobs documentation</a> .
Functions	See <a href="#">Continuous deployment for Azure Functions</a> .

To customize your deployment, include a `.deployment` file in the repository root. For more information, see [Customize deployments](#) and [Custom deployment script](#).

**NOTE**

If you use Visual Studio, let [Visual Studio create a repository for you](#). Your project will immediately be ready for deployment via Git.

## Configure a deployment user

See [Configure deployment credentials for Azure App Service](#). You can use either user-scope credentials or application-scope credentials.

## Create a Git enabled app

If you already have an App Service app and want to configure local Git deployment for it, see [Configure an existing app](#) instead.

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Azure portal](#)

Run `az webapp create` with the `--deployment-local-git` option. For example:

```
az webapp create --resource-group <group-name> --plan <plan-name> --name <app-name> --runtime "<runtime-flag>" --deployment-local-git
```

The output contains a URL like: `https://<deployment-username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Use this URL to deploy your app in the next step.

## Configure an existing app

If you haven't created an app yet, see [Create a Git enabled app](#) instead.

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Azure portal](#)

Run `az webapp deployment source config-local-git`. For example:

```
az webapp deployment source config-local-git --name <app-name> --resource-group <group-name>
```

The output contains a URL like: `https://<deployment-username>@<app-name>.scm.azurewebsites.net/<app-name>.git`. Use this URL to deploy your app in the next step.

**TIP**

This URL contains the user-scope deployment username. If you like, you can [use the application-scope credentials](#) instead.

## Deploy the web app

1. In a local terminal window, change the directory to the root of your Git repository, and add a Git remote using the URL you got from your app. If your chosen method doesn't give you a URL, use `https://<app-name>.scm.azurewebsites.net/<app-name>.git` with your app name in `<app-name>`.

```
git remote add azure <url>
```

#### NOTE

If you [created a Git-enabled app in PowerShell using New-AzWebApp](#), the remote is already created for you.

2. Push to the Azure remote with `git push azure master` (see [Change deployment branch](#)).
3. In the **Git Credential Manager** window, enter your [user-scope or application-scope credentials](#), not your Azure sign-in credentials.  
If your Git remote URL already contains the username and password, you won't be prompted.
4. Review the output. You may see runtime-specific automation, such as MSBuild for ASP.NET, `npm install` for Node.js, and `pip install` for Python.
5. Browse to your app in the Azure portal to verify that the content is deployed.

## Change deployment branch

When you push commits to your App Service repository, App Service deploys the files in the `master` branch by default. Because many Git repositories are moving away from `master` to `main`, you need to make sure that you push to the right branch in the App Service repository in one of two ways:

- Deploy to `master` explicitly with a command like:

```
git push azure main:master
```

- Change the deployment branch by setting the `DEPLOYMENT_BRANCH` app setting, then push commits to the custom branch. To do it with Azure CLI:

```
az webapp config appsettings set --name <app-name> --resource-group <group-name> --settings
DEPLOYMENT_BRANCH='main'
git push azure main
```

## Troubleshoot deployment

You may see the following common error messages when you use Git to publish to an App Service app in Azure:

MESSAGE	CAUSE	RESOLUTION
<code>Unable to access '[siteURL]': Failed to connect to [scmAddress]</code>	The app isn't up and running.	Start the app in the Azure portal. Git deployment isn't available when the web app is stopped.
<code>Couldn't resolve host 'hostname'</code>	The address information for the 'azure' remote is incorrect.	Use the <code>git remote -v</code> command to list all remotes, along with the associated URL. Verify that the URL for the 'azure' remote is correct. If needed, remove and recreate this remote using the correct URL.

MESSAGE	CAUSE	RESOLUTION
No refs in common and none specified; doing nothing. Perhaps you should specify a branch such as 'main'.	You didn't specify a branch during <code>git push</code> , or you haven't set the <code>push.default</code> value in <code>.gitconfig</code> .	Run <code>git push</code> again, specifying the main branch: <code>git push azure main</code> .
Error - Changes committed to remote repository but deployment to website failed.	You pushed a local branch that doesn't match the app deployment branch on 'azure'.	Verify that current branch is <code>master</code> . To change the default branch, use <code>DEPLOYMENT_BRANCH</code> application setting (see <a href="#">Change deployment branch</a> ).
<code>src refspec [branchname] does not match any.</code>	You tried to push to a branch other than main on the 'azure' remote.	Run <code>git push</code> again, specifying the main branch: <code>git push azure main</code> .
<code>RPC failed; result=22, HTTP code = 5xx.</code>	This error can happen if you try to push a large git repository over HTTPS.	Change the git configuration on the local machine to make the <code>postBuffer</code> bigger. For example: <code>git config --global http.postBuffer 524288000</code>
Error - Changes committed to remote repository but your web app not updated.	You deployed a Node.js app with a <code>package.json</code> file that specifies additional required modules.	Review the <code>npm ERR!</code> error messages before this error for more context on the failure. The following are the known causes of this error, and the corresponding <code>npm ERR!</code> messages:  <b>Malformed package.json file:</b> <code>npm ERR! Couldn't read dependencies.</code>  <b>Native module doesn't have a binary distribution for Windows:</b> <code>npm ERR! \cmd "/c" "node-gyp rebuild"\ failed with 1</code> or <code>npm ERR! [modulename@version] preinstall: \make    gmake\</code>

## Additional resources

- [App Service build server \(Project Kudu documentation\)](#)
- [Continuous deployment to Azure App Service](#)
- [Sample: Create a web app and deploy code from a local Git repository \(Azure CLI\)](#)
- [Sample: Create a web app and deploy code from a local Git repository \(PowerShell\)](#)

# Deploy to App Service using GitHub Actions

11/2/2021 • 13 minutes to read • [Edit Online](#)

Get started with [GitHub Actions](#) to automate your workflow and deploy to [Azure App Service](#) from GitHub.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- A GitHub account. If you don't have one, sign up for [free](#).
- A working Azure App Service app.
  - .NET: [Create an ASP.NET Core web app in Azure](#)
  - ASP.NET: [Create an ASP.NET Framework web app in Azure](#)
  - JavaScript: [Create a Node.js web app in Azure App Service](#)
  - Java: [Create a Java app on Azure App Service](#)
  - Python: [Create a Python app in Azure App Service](#)

## Workflow file overview

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that make up the workflow.

The file has three sections:

SECTION	TASKS
Authentication	1. Define a service principal or publish profile. 2. Create a GitHub secret.
Build	1. Set up the environment. 2. Build the web app.
Deploy	1. Deploy the web app.

## Use the Deployment Center

You can quickly get started with GitHub Actions by using the App Service Deployment Center. This will automatically generate a workflow file based on your application stack and commit it to your GitHub repository in the correct directory.

1. Navigate to your webapp in the Azure portal
2. On the left side, click **Deployment Center**
3. Under **Continuous Deployment (CI / CD)**, select **GitHub**
4. Next, select **GitHub Actions**
5. Use the dropdowns to select your GitHub repository, branch, and application stack
  - If the selected branch is protected, you can still continue to add the workflow file. Be sure to review your branch protections before continuing.
6. On the final screen, you can review your selections and preview the workflow file that will be committed to the repository. If the selections are correct, click **Finish**

This will commit the workflow file to the repository. The workflow to build and deploy your app will start immediately.

## Set up a workflow manually

You can also deploy a workflow without using the Deployment Center. To do so, you will need to first generate deployment credentials.

## Generate deployment credentials

The recommended way to authenticate with Azure App Services for GitHub Actions is with a publish profile. You can also authenticate with a service principal but the process requires more steps.

Save your publish profile credential or service principal as a [GitHub secret](#) to authenticate with Azure. You'll access the secret within your workflow.

- [Publish profile](#)
- [Service principal](#)

A publish profile is an app-level credential. Set up your publish profile as a GitHub secret.

1. Go to your app service in the Azure portal.
2. On the **Overview** page, select **Get Publish profile**.
3. Save the downloaded file. You'll use the contents of the file to create a GitHub secret.

### NOTE

As of October 2020, Linux web apps will need the app setting `WEBSITE_WEBDEPLOY_USE_SCM` set to `true` before downloading the publish profile. This requirement will be removed in the future.

## Configure the GitHub secret

- [Publish profile](#)
- [Service principal](#)

In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**.

To use [app-level credentials](#), paste the contents of the downloaded publish profile file into the secret's value field. Name the secret `AZURE_WEBAPP_PUBLISH_PROFILE`.

When you configure your GitHub workflow, you use the `AZURE_WEBAPP_PUBLISH_PROFILE` in the deploy Azure Web App action. For example:

```
- uses: azure/webapps-deploy@v2
 with:
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
```

## Set up the environment

Setting up the environment can be done using one of the setup actions.

LANGUAGE	SETUP ACTION
.NET	<code>actions/setup-dotnet</code>
ASP.NET	<code>actions/setup-dotnet</code>
Java	<code>actions/setup-java</code>
JavaScript	<code>actions/setup-node</code>
Python	<code>actions/setup-python</code>

The following examples show how to set up the environment for the different supported languages:

### .NET

```
- name: Setup Dotnet 3.3.x
 uses: actions/setup-dotnet@v1
 with:
 dotnet-version: '3.3.x'
```

### ASP.NET

```
- name: Install Nuget
 uses: nuget/setup-nuget@v1
 with:
 nuget-version: ${{ env.NUGET_VERSION }}
```

### Java

```
- name: Setup Java 1.8.x
 uses: actions/setup-java@v1
 with:
 # If your pom.xml <maven.compiler.source> version is not in 1.8.x,
 # change the Java version to match the version in pom.xml <maven.compiler.source>
 java-version: '1.8.x'
```

### JavaScript

```
env:
 NODE_VERSION: '14.x' # set this to the node version to use

jobs:
 build-and-deploy:
 name: Build and Deploy
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@main
 - name: Use Node.js ${{ env.NODE_VERSION }}
 uses: actions/setup-node@v1
 with:
 node-version: ${{ env.NODE_VERSION }}
```

### Python

```
- name: Setup Python 3.x
uses: actions/setup-python@v1
with:
 python-version: 3.x
```

## Build the web app

The process of building a web app and deploying to Azure App Service changes depending on the language.

The following examples show the part of the workflow that builds the web app, in different supported languages.

For all languages, you can set the web app root directory with `working-directory`.

### .NET

The environment variable `AZURE_WEBAPP_PACKAGE_PATH` sets the path to your web app project.

```
- name: dotnet build and publish
run: |
 dotnet restore
 dotnet build --configuration Release
 dotnet publish -c Release -o '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'
```

### ASP.NET

You can restore NuGet dependencies and run msbuild with `run`.

```
- name: NuGet to restore dependencies as well as project-specific tools that are specified in the project file
run: nuget restore

- name: Add msbuild to PATH
uses: microsoft/setup-msbuild@v1.0.2

- name: Run msbuild
run: msbuild .\SampleWebApplication.sln
```

### Java

```
- name: Build with Maven
run: mvn package --file pom.xml
```

### JavaScript

For Node.js, you can set `working-directory` or change for npm directory in `pushd`.

```
- name: npm install, build, and test
run: |
 npm install
 npm run build --if-present
 npm run test --if-present
 working-directory: my-app-folder # set to the folder with your app if it is not the root directory
```

### Python

```
- name: Install dependencies
 run: |
 python -m pip install --upgrade pip
 pip install -r requirements.txt
```

## Deploy to App Service

To deploy your code to an App Service app, use the `azure/webapps-deploy@v2` action. This action has four parameters:

PARAMETER	EXPLANATION
<code>app-name</code>	(Required) Name of the App Service app
<code>publish-profile</code>	(Optional) Publish profile file contents with Web Deploy secrets
<code>package</code>	(Optional) Path to package or folder. The path can include *.zip, *.war, *.jar, or a folder to deploy
<code>slot-name</code>	(Optional) Enter an existing slot other than the production slot

- [Publish profile](#)
- [Service principal](#)

### .NET Core

Build and deploy a .NET Core app to Azure using an Azure publish profile. The `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

```
name: .NET Core CI

on: [push]

env:
 AZURE_WEBAPP_NAME: my-app-name # set this to your application's name
 AZURE_WEBAPP_PACKAGE_PATH: '.' # set this to the path to your web app project, defaults to the
repository root
 DOTNET_VERSION: '3.1.x' # set this to the dot net version to use

jobs:
 build:
 runs-on: ubuntu-latest

 steps:
 # Checkout the repo
 - uses: actions/checkout@main

 # Setup .NET Core SDK
 - name: Setup .NET Core
 uses: actions/setup-dotnet@v1
 with:
 dotnet-version: ${{ env.DOTNET_VERSION }}

 # Run dotnet build and publish
 - name: dotnet build and publish
 run: |
 dotnet restore
 dotnet build --configuration Release
 dotnet publish -c Release -o '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'

 # Deploy to Azure Web apps
 - name: 'Run Azure webapp deploy action using publish profile credentials'
 uses: azure/webapps-deploy@v2
 with:
 app-name: '${{ env.AZURE_WEBAPP_NAME }}' # Replace with your app name
 publish-profile: '${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}' # Define secret variable in
repository settings as per action documentation
 package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/myapp'
```

## ASP.NET

Build and deploy an ASP.NET MVC app that uses NuGet and `publish-profile` for authentication.

```

name: Deploy ASP.NET MVC App deploy to Azure Web App

on: [push]

env:
 AZURE_WEBAPP_NAME: my-app # set this to your application's name
 AZURE_WEBAPP_PACKAGE_PATH: '.' # set this to the path to your web app project, defaults to the
repository root
 NUGET_VERSION: '5.3.x' # set this to the dot net version to use

jobs:
 build-and-deploy:
 runs-on: windows-latest
 steps:
 - uses: actions/checkout@main

 - name: Install Nuget
 uses: nuget/setup-nuget@v1
 with:
 nuget-version: ${{ env.NUGET_VERSION}}
 - name: NuGet to restore dependencies as well as project-specific tools that are specified in the
project file
 run: nuget restore

 - name: Add msbuild to PATH
 uses: microsoft/setup-msbuild@v1.0.2

 - name: Run MSBuild
 run: msbuild .\SampleWebApplication.sln

 - name: 'Run Azure webapp deploy action using publish profile credentials'
 uses: azure/webapps-deploy@v2
 with:
 app-name: ${{ env.AZURE_WEBAPP_NAME }} # Replace with your app name
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }} # Define secret variable in repository
settings as per action documentation
 package: '${{ env.AZURE_WEBAPP_PACKAGE_PATH }}/SampleWebApplication/'

```

## Java

Build and deploy a Java Spring app to Azure using an Azure publish profile. The `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

```
name: Java CI with Maven

on: [push]

jobs:
 build:

 runs-on: ubuntu-latest

 steps:
 - uses: actions/checkout@v2
 - name: Set up JDK 1.8
 uses: actions/setup-java@v1
 with:
 java-version: 1.8
 - name: Build with Maven
 run: mvn -B package --file pom.xml
 working-directory: my-app-path
 - name: Azure WebApp
 uses: Azure/webapps-deploy@v2
 with:
 app-name: my-app-name
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
 package: my/target/*.jar
```

To deploy a `war` instead of a `jar`, change the `package` value.

```
- name: Azure WebApp
uses: Azure/webapps-deploy@v2
with:
 app-name: my-app-name
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
 package: my/target/*.war
```

## JavaScript

Build and deploy a Node.js app to Azure using the app's publish profile. The `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

```

File: .github/workflows/workflow.yml
name: JavaScript CI

on: [push]

env:
 AZURE_WEBAPP_NAME: my-app-name # set this to your application's name
 AZURE_WEBAPP_PACKAGE_PATH: 'my-app-path' # set this to the path to your web app project, defaults to
the repository root
 NODE_VERSION: '14.x' # set this to the node version to use

jobs:
 build-and-deploy:
 name: Build and Deploy
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@main
 - name: Use Node.js ${{ env.NODE_VERSION }}
 uses: actions/setup-node@v1
 with:
 node-version: ${{ env.NODE_VERSION }}
 - name: npm install, build, and test
 run: |
 # Build and test the project, then
 # deploy to Azure Web App.
 npm install
 npm run build --if-present
 npm run test --if-present
 working-directory: my-app-path
 - name: 'Deploy to Azure WebApp'
 uses: azure/webapps-deploy@v2
 with:
 app-name: ${{ env.AZURE_WEBAPP_NAME }}
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
 package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}

```

## Python

Build and deploy a Python app to Azure using the app's publish profile. Note how the `publish-profile` input references the `AZURE_WEBAPP_PUBLISH_PROFILE` secret that you created earlier.

```
name: Python CI

on:
 [push]

env:
 AZURE_WEBAPP_NAME: my-web-app # set this to your application's name
 AZURE_WEBAPP_PACKAGE_PATH: '.' # set this to the path to your web app project, defaults to the repository root

jobs:
 build:
 runs-on: ubuntu-latest
 steps:
 - uses: actions/checkout@v2
 - name: Set up Python 3.x
 uses: actions/setup-python@v2
 with:
 python-version: 3.x
 - name: Install dependencies
 run: |
 python -m pip install --upgrade pip
 pip install -r requirements.txt
 - name: Building web app
 uses: azure/appservice-build@v2
 - name: Deploy web App using GH Action azure/webapps-deploy
 uses: azure/webapps-deploy@v2
 with:
 app-name: ${{ env.AZURE_WEBAPP_NAME }}
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
 package: ${{ env.AZURE_WEBAPP_PACKAGE_PATH }}
```

## Next steps

You can find our set of Actions grouped into different repositories on GitHub, each one containing documentation and examples to help you use GitHub for CI/CD and deploy your apps to Azure.

- [Actions workflows to deploy to Azure](#)
- [Azure login](#)
- [Azure WebApp](#)
- [Azure WebApp for containers](#)
- [Docker login/logout](#)
- [Events that trigger workflows](#)
- [K8s deploy](#)
- [Starter Workflows](#)

# Deploy a custom container to App Service using GitHub Actions

11/2/2021 • 6 minutes to read • [Edit Online](#)

[GitHub Actions](#) gives you the flexibility to build an automated software development workflow. With the [Azure Web Deploy action](#), you can automate your workflow to deploy custom containers to [App Service](#) using GitHub Actions.

A workflow is defined by a YAML (.yml) file in the `/.github/workflows/` path in your repository. This definition contains the various steps and parameters that are in the workflow.

For an Azure App Service container workflow, the file has three sections:

SECTION	TASKS
Authentication	1. Retrieve a service principal or publish profile. 2. Create a GitHub secret.
Build	1. Create the environment. 2. Build the container image.
Deploy	1. Deploy the container image.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#)
- A GitHub account. If you don't have one, sign up for [free](#). You need to have code in a GitHub repository to deploy to Azure App Service.
- A working container registry and Azure App Service app for containers. This example uses Azure Container Registry. Make sure to complete the full deployment to Azure App Service for containers. Unlike regular web apps, web apps for containers do not have a default landing page. Publish the container to have a working example.
  - [Learn how to create a containerized Node.js application using Docker, push the container image to a registry, and then deploy the image to Azure App Service](#)

## Generate deployment credentials

The recommended way to authenticate with Azure App Services for GitHub Actions is with a publish profile. You can also authenticate with a service principal but the process requires more steps.

Save your publish profile credential or service principal as a [GitHub secret](#) to authenticate with Azure. You'll access the secret within your workflow.

- [Publish profile](#)
- [Service principal](#)

A publish profile is an app-level credential. Set up your publish profile as a GitHub secret.

1. Go to your app service in the Azure portal.

2. On the **Overview** page, select **Get Publish profile**.

**NOTE**

As of October 2020, Linux web apps will need the app setting `WEBSITE_WEBDEPLOY_USE_SCM` set to `true` before downloading the file. This requirement will be removed in the future. See [Configure an App Service app in the Azure portal](#), to learn how to configure common web app settings.

3. Save the downloaded file. You'll use the contents of the file to create a GitHub secret.

## Configure the GitHub secret for authentication

- [Publish profile](#)
- [Service principal](#)

In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**.

To use [app-level credentials](#), paste the contents of the downloaded publish profile file into the secret's value field. Name the secret `AZURE_WEBAPP_PUBLISH_PROFILE`.

When you configure your GitHub workflow, you use the `AZURE_WEBAPP_PUBLISH_PROFILE` in the deploy Azure Web App action. For example:

```
- uses: azure/webapps-deploy@v2
 with:
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
```

## Configure GitHub secrets for your registry

Define secrets to use with the Docker Login action. The example in this document uses Azure Container Registry for the container registry.

1. Go to your container in the Azure portal or Docker and copy the username and password. You can find the Azure Container Registry username and password in the Azure portal under **Settings > Access keys** for your registry.
2. Define a new secret for the registry username named `REGISTRY_USERNAME`.
3. Define a new secret for the registry password named `REGISTRY_PASSWORD`.

## Build the Container image

The following example show part of the workflow that builds a NodeJS Docker image. Use [Docker Login](#) to log into a private container registry. This example uses Azure Container Registry but the same action works for other registries.

```

name: Linux Container Node Workflow

on: [push]

jobs:
 build:
 runs-on: ubuntu-latest

 steps:
 - uses: actions/checkout@v2
 - uses: azure/docker-login@v1
 with:
 login-server: mycontainer.azurecr.io
 username: ${{ secrets.REGISTRY_USERNAME }}
 password: ${{ secrets.REGISTRY_PASSWORD }}
 - run: |
 docker build . -t mycontainer.azurecr.io/myapp:${{ github.sha }}
 docker push mycontainer.azurecr.io/myapp:${{ github.sha }}

```

You can also use [Docker Login](#) to log into multiple container registries at the same time. This example includes two new GitHub secrets for authentication with docker.io. The example assumes that there is a Dockerfile at the root level of the registry.

```

name: Linux Container Node Workflow

on: [push]

jobs:
 build:
 runs-on: ubuntu-latest

 steps:
 - uses: actions/checkout@v2
 - uses: azure/docker-login@v1
 with:
 login-server: mycontainer.azurecr.io
 username: ${{ secrets.REGISTRY_USERNAME }}
 password: ${{ secrets.REGISTRY_PASSWORD }}
 - uses: azure/docker-login@v1
 with:
 login-server: index.docker.io
 username: ${{ secrets.DOCKERIO_USERNAME }}
 password: ${{ secrets.DOCKERIO_PASSWORD }}
 - run: |
 docker build . -t mycontainer.azurecr.io/myapp:${{ github.sha }}
 docker push mycontainer.azurecr.io/myapp:${{ github.sha }}

```

## Deploy to an App Service container

To deploy your image to a custom container in App Service, use the `azure/webapps-deploy@v2` action. This action has seven parameters:

PARAMETER	EXPLANATION
<b>app-name</b>	(Required) Name of the App Service app
<b>publish-profile</b>	(Optional) Applies to Web Apps(Windows and Linux) and Web App Containers(linux). Multi container scenario not supported. Publish profile (*.publishsettings) file contents with Web Deploy secrets

PARAMETER	EXPLANATION
<b>slot-name</b>	(Optional) Enter an existing Slot other than the Production slot
<b>package</b>	(Optional) Applies to Web App only: Path to package or folder. *.zip, *.war, *.jar or a folder to deploy
<b>images</b>	(Required) Applies to Web App Containers only: Specify the fully qualified container image(s) name. For example, 'myregistry.azurecr.io/nginx:latest' or 'python:3.7.2-alpine/'. For a multi-container app, multiple container image names can be provided (multi-line separated)
<b>configuration-file</b>	(Optional) Applies to Web App Containers only: Path of the Docker-Compose file. Should be a fully qualified path or relative to the default working directory. Required for multi-container apps.
<b>startup-command</b>	(Optional) Enter the start-up command. For ex. dotnet run or dotnet filename.dll

- [Publish profile](#)
- [Service principal](#)

```

name: Linux Container Node Workflow

on: [push]

jobs:
 build:
 runs-on: ubuntu-latest

 steps:
 - uses: actions/checkout@v2

 - uses: azure/docker-login@v1
 with:
 login-server: mycontainer.azurecr.io
 username: ${{ secrets.REGISTRY_USERNAME }}
 password: ${{ secrets.REGISTRY_PASSWORD }}

 - run: |
 docker build . -t mycontainer.azurecr.io/myapp:${{ github.sha }}
 docker push mycontainer.azurecr.io/myapp:${{ github.sha }}

 - uses: azure/webapps-deploy@v2
 with:
 app-name: 'myapp'
 publish-profile: ${{ secrets.AZURE_WEBAPP_PUBLISH_PROFILE }}
 images: 'mycontainer.azurecr.io/myapp:${{ github.sha }}'
```

## Next steps

You can find our set of Actions grouped into different repositories on GitHub, each one containing documentation and examples to help you use GitHub for CI/CD and deploy your apps to Azure.

- [Actions workflows to deploy to Azure](#)

- Azure login
- Azure WebApp
- Docker login/logout
- Events that trigger workflows
- K8s deploy
- Starter Workflows

# Provision and deploy microservices predictably in Azure

11/2/2021 • 14 minutes to read • [Edit Online](#)

This tutorial shows how to provision and deploy an application composed of [microservices](#) in [Azure App Service](#) as a single unit and in a predictable manner using JSON resource group templates and PowerShell scripting.

When provisioning and deploying high-scale applications that are composed of highly decoupled microservices, repeatability and predictability are crucial to success. [Azure App Service](#) enables you to create microservices that include web apps, mobile back ends, and API apps. [Azure Resource Manager](#) enables you to manage all the microservices as a unit, together with resource dependencies such as database and source control settings. Now, you can also deploy such an application using JSON templates and simple PowerShell scripting.

## What you will do

In the tutorial, you will deploy an application that includes:

- Two App Service apps (i.e. two microservices)
- A backend SQL Database
- App settings, connection strings, and source control
- Application insights, alerts, autoscaling settings

## Tools you will use

In this tutorial, you will use the following tools. Since it's not comprehensive discussion on tools, I'm going to stick to the end-to-end scenario and just give you a brief intro to each, and where you can find more information on it.

### Azure Resource Manager templates (JSON)

Every time you create an app in Azure App Service, for example, Azure Resource Manager uses a JSON template to create the entire resource group with the component resources. A complex template from the [Azure Marketplace](#) can include the database, storage accounts, the App Service plan, the app itself, alert rules, app settings, autoscale settings, and more, and all these templates are available to you through PowerShell. For more information on the Azure Resource Manager templates, see [Authoring Azure Resource Manager templates](#)

### Azure SDK 2.6 for Visual Studio

The newest SDK contains improvements to the Resource Manager template support in the JSON editor. You can use this to quickly create a resource group template from scratch or open an existing JSON template (such as a downloaded gallery template) for modification, populate the parameters file, and even deploy the resource group directly from an Azure Resource Group solution.

For more information, see [Azure SDK 2.6 for Visual Studio](#).

### Azure PowerShell 0.8.0 or later

Beginning in version 0.8.0, the Azure PowerShell installation includes the Azure Resource Manager module in addition to the Azure module. This new module enables you to script the deployment of resource groups.

For more information, see [Using Azure PowerShell with Azure Resource Manager](#)

### Azure Resource Explorer

This [preview tool](#) enables you to explore the JSON definitions of all the resource groups in your subscription and the individual resources. In the tool, you can edit the JSON definitions of a resource, delete an entire hierarchy of resources, and create new resources. The information readily available in this tool is very helpful for template authoring because it shows you what properties you need to set for a particular type of resource, the correct values, etc. You can even create your resource group in the [Azure Portal](#), then inspect its JSON definitions in the explorer tool to help you template the resource group.

### Deploy to Azure button

If you use GitHub for source control, you can put a [Deploy to Azure button](#) into your README.MD, which enables a turn-key deployment UI to Azure. While you can do this for any simple app, you can extend this to enable deploying an entire resource group by putting an azuredeploy.json file in the repository root. This JSON file, which contains the resource group template, will be used by the Deploy to Azure button to create the resource group. For an example, see the [ToDoApp](#) sample, which you will use in this tutorial.

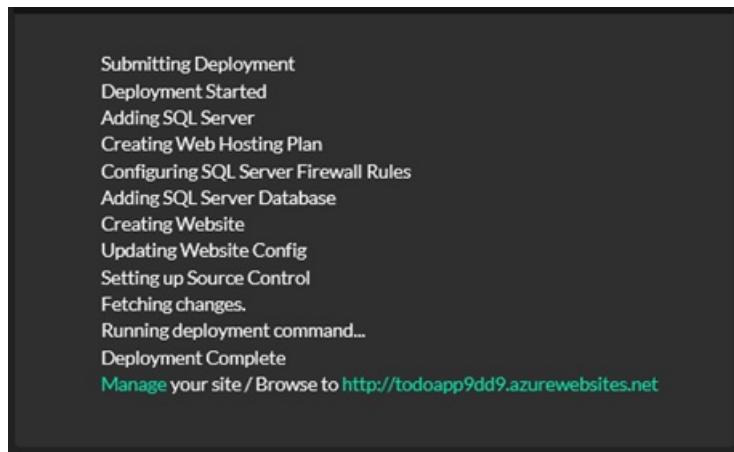
## Get the sample resource group template

So now let's get right to it.

1. Navigate to the [ToDoApp](#) App Service sample.
2. In readme.md, click **Deploy to Azure**.
3. You're taken to the [deploy-to-azure](#) site and asked to input deployment parameters. Notice that most of the fields are populated with the repository name and some random strings for you. You can change all the fields if you want, but the only things you have to enter are the SQL Server administrative login and the password, then click **Next**.

The screenshot shows a deployment configuration dialog. At the top, it displays the Git Repository URL as <https://github.com/Azure-AppService-Samples/ToDoApp> and the Branch as master. Below this, there are two main sections: 'Directory' and 'Subscription'. Under 'Directory', 'Default Directory' is selected. Under 'Subscription', 'Visual Studio Ultimate with MSDN' is selected. The 'Resource Group' section shows 'Create New' selected and the name 'ToDoApp9dd9' entered. The 'Site' section includes 'Site Name' (ToDoApp9dd9), 'Site Location' (Brazil South), and 'Sku' (Free). The 'Sql' section includes 'Sql Server Name' (todoapp9dd9-server), 'Sql Server Location' (East US 2), 'Sql Server Admin Login' (empty), 'Sql Db Name' (DemosDB), 'Sql Db Collation' (SQL\_Latin1\_General\_CI\_AS), 'Sql Db Edition' (Web), 'Sql Db Max Size Bytes' (1073741824), and 'Sql Db Service Objective Id' (910b4fcf-8a29-4c3e-958f-f7ba794388b2). At the bottom right is a blue 'Next' button.

4. Next, click **Deploy** to start the deployment process. Once the process runs to completion, click the <http://todoappXXXX.azurewebsites.net> link to browse the deployed application.



5. Back in the Deploy page, click the **Manage** link to see the new application in the Azure Portal.
6. In the **Essentials** dropdown, click the resource group link. Note also that the app is already connected to the GitHub repository under **External Project**.

RUNNING  
todoapp9dd9  
WEB APP

Settings    Browse    Start    Stop    Swap    Restart    Delete    Get publish...

Essentials ^

Resource group <b>ToDoApp9dd9</b>	URL <a href="http://todoapp9dd9.azurewebsites.net">http://todoapp9dd9.azurewebsites.net</a>
Status Running	App Service plan/pricing tier <b>ToDoApp9dd9 (Free)</b>
Location Brazil South	External Project <b>ToDoApp</b>
Subscription name	

7. In the resource group blade, note that there are already two apps and one SQL Database in the resource group.

Summary    Add tiles (+)

Resources

todoappb8f7-server
DemosDB
ToDoAppb8f7
ToDoAppb8f7
ToDoAppb8f7Api

Everything that you just saw in a few short minutes is a fully deployed two-microservice application, with all the components, dependencies, settings, database, and continuous publishing, set up by an automated orchestration in Azure Resource Manager. All this was done by two things:

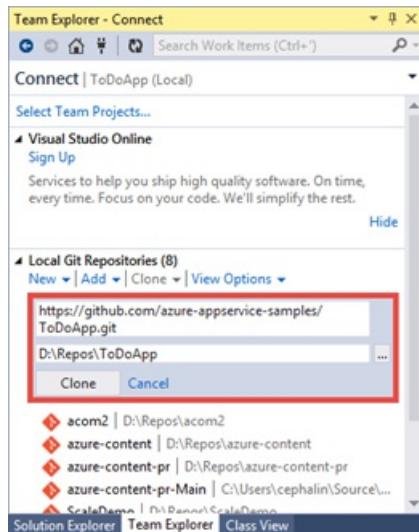
- The Deploy to Azure button
- `azuredeploy.json` in the repo root

You can deploy this same application tens, hundreds, or thousands of times and have the exact same configuration every time. The repeatability and the predictability of this approach enables you to deploy high-scale applications with ease and confidence.

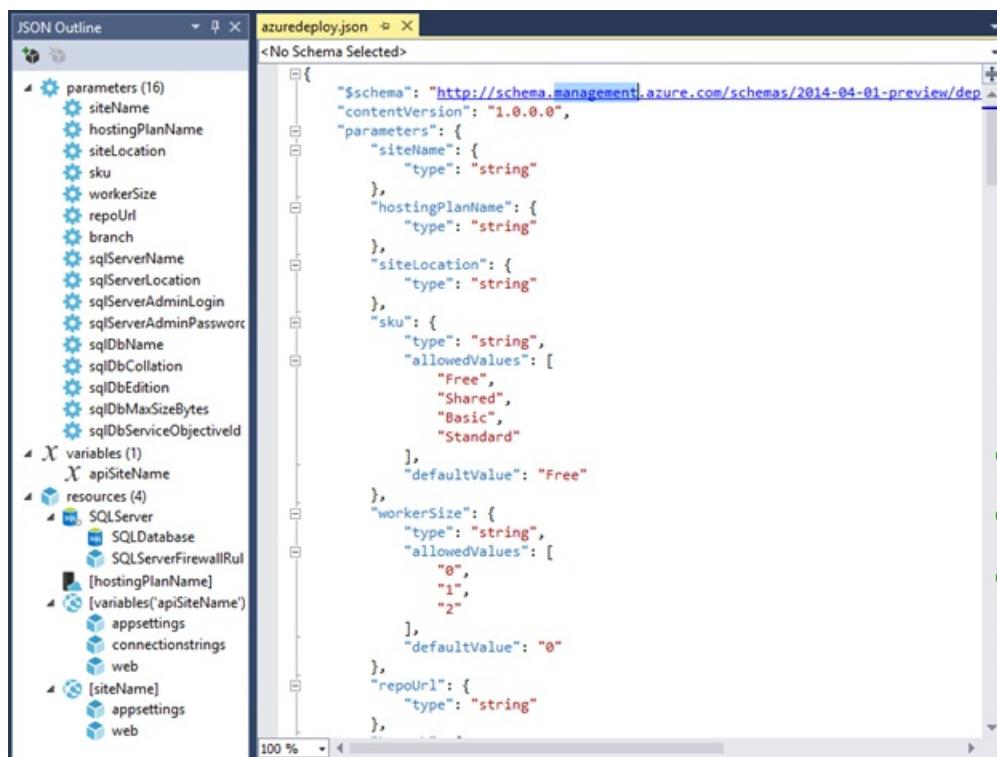
## Examine (or edit) AZUREDEPLOY.JSON

Now let's look at how the GitHub repository was set up. You will be using the JSON editor in the Azure .NET SDK, so if you haven't already installed [Azure .NET SDK 2.6](#), do it now.

1. Clone the [ToDoApp](#) repository using your favorite git tool. In the screenshot below, I'm doing this in the Team Explorer in Visual Studio 2013.



2. From the repository root, open azuredeploy.json in Visual Studio. If you don't see the JSON Outline pane, you need to install Azure .NET SDK.



I'm not going to describe every detail of the JSON format, but the [More Resources](#) section has links for learning the resource group template language. Here, I'm just going to show you the interesting features that can help you get started in making your own custom template for app deployment.

### Parameters

Take a look at the parameters section to see that most of these parameters are what the Deploy to Azure button prompts you to input. The site behind the Deploy to Azure button populates the input UI using the parameters defined in azureddeploy.json. These parameters are used throughout the resource definitions, such as resource names, property values, etc.

## Resources

In the resources node, you can see that 4 top-level resources are defined, including a SQL Server instance, an App Service plan, and two apps.

### App Service plan

Let's start with a simple root-level resource in the JSON. In the JSON Outline, click the App Service plan named [hostingPlanName] to highlight the corresponding JSON code.



```
JSON Outline azureddeploy.json http://schema.management.azure.com/schemas/2014-04-01-preview/deploymentTemplate.json

parameters (16)
variables (1)
resources (4)
 SQLServer
 SQLDatabase
 SQLServerFirewallRule
 [hostingPlanName]
 [variables('apiSiteName')]
 appsettings
 connectionstrings
 web
 [siteName]
 appsettings
 web

{
 "apiVersion": "2014-11-01",
 "name": "[parameters('hostingPlanName')]",
 "type": "Microsoft.Web/serverFarms",
 "location": "[parameters('siteLocation')]",
 "properties": {
 "name": "[parameters('hostingPlanName')]",
 "sku": "[parameters('sku')]",
 "workerSize": "[parameters('workerSize')]",
 "numberOfWorkers": 1
 }
},
{
 "apiVersion": "2015-04-01",
 "name": "[variables('apiSiteName')]",
 "type": "Microsoft.Web/sites",
 ...
}
```

Note that the `type` element specifies the string for an App Service plan (it was called a server farm a long, long time ago), and other elements and properties are filled in using the parameters defined in the JSON file, and this resource doesn't have any nested resources.

#### NOTE

Note also that the value of `apiVersion` tells Azure which version of the REST API to use the JSON resource definition with, and it can affect how the resource should be formatted inside the `{ }` .

## SQL Server

Next, click on the SQL Server resource named `SQLServer` in the JSON Outline.

```

{
 "apiVersion": "2014-04-01-preview",
 "name": "[parameters('sqlServerName')]",
 "type": "Microsoft.Sql/servers",
 "location": "[parameters('sqlServerLocation')]",
 "tags": {
 "displayName": "SQLServer"
 },
 "properties": {
 "administratorLogin": "[parameters('sqlServerAdminLogin')]",
 "administratorLoginPassword": "[parameters('sqlServerAdminPassword')]"
 },
 "resources": [
 {
 "apiVersion": "2014-11-01",
 "name": "[parameters('sqlDbName')]",
 "type": "databases",
 "location": "[parameters('sqlServerLocation')]",
 "tags": {
 "displayName": "SQLDatabase"
 },
 "dependsOn": [
 "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
],
 "properties": {
 "edition": "[parameters('sqlDbEdition')]",
 "collation": "[parameters('sqlDbCollation')]",
 "maxSizeBytes": "[parameters('sqlDbMaxSizeBytes')]",
 "requestedServiceObjectiveId": "[parameters('sqlDbServiceObjectiveId')]"
 }
 },
 {
 "apiVersion": "2014-11-01",
 "name": "SQLServerFirewallRules",
 "type": "firewallrules",
 "location": "[parameters('sqlServerLocation')]",
 "dependsOn": [
 "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
],
 "properties": {
 "endIpAddress": "0.0.0.0",
 "startIpAddress": "0.0.0.0"
 }
 }
]
}

```

Note the following about the highlighted JSON code:

- The use of parameters ensures that the created resources are named and configured in a way that makes them consistent with one another.
- The SQLServer resource has two nested resources, each has a different value for `type`.
- The nested resources inside `"resources": [...]`, where the database and the firewall rules are defined, have a `dependsOn` element that specifies the resource ID of the root-level SQLServer resource. This tells Azure Resource Manager, “before you create this resource, that other resource must already exist; and if that other resource is defined in the template, then create that one first”.

#### NOTE

For detailed information on how to use the `resourceId()` function, see [Azure Resource Manager Template Functions](#).

- The effect of the `dependsOn` element is that Azure Resource Manager can know which resources can be created in parallel and which resources must be created sequentially.

#### App Service app

Now, let's move on to the actual apps themselves, which are more complicated. Click the `[variables('apiSiteName')]` app in the JSON Outline to highlight its JSON code. You'll notice that things are getting much more interesting. For this purpose, I'll talk about the features one by one:

##### Root resource

The app depends on two different resources. This means that Azure Resource Manager will create the app only after both the App Service plan and the SQL Server instance are created.

```

"dependsOn": [
 "[resourceId('Microsoft.Web/serverFarms', parameters('hostingPlanName'))]",
 "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
],

```

##### App settings

The app settings are also defined as a nested resource.

```
{
 "apiVersion": "2015-04-01",
 "name": "appsettings",
 "type": "config",
 "dependsOn": [
 "[resourceId('Microsoft.Web/Sites', variables('apiSiteName'))]"
],
 "properties": {
 "PROJECT": "src\\MultiChannelToDo\\MultiChannelToDo.csproj",
 "clientUrl": "[concat('http://', parameters('siteName'), '.azurewebsites.net')]"
 }
},
```

In the `properties` element for `config/appsettings`, you have two app settings in the format `"<name>" : "<value>"`.

- `PROJECT` is a [KUDU setting](#) that tells Azure deployment which project to use in a multi-project Visual Studio solution. I will show you later how source control is configured, but since the ToDoApp code is in a multi-project Visual Studio solution, we need this setting.
- `clientUrl` is simply an app setting that the application code uses.

#### Connection strings

The connection strings are also defined as a nested resource.

```
{
 "apiVersion": "2015-04-01",
 "name": "connectionstrings",
 "type": "config",
 "dependsOn": [
 "[resourceId('Microsoft.Web/Sites', variables('apiSiteName'))]",
 "[resourceId('Microsoft.Sql/servers', parameters('sqlServerName'))]"
],
 "properties": {
 "MultiChannelToDoContext": { "value": "[concat('Data Source=tcp:', reference(concat('M", "ultiChannelToDoContext', 'sqlServerName')), '.', parameters('apiDatabaseName'))]" }
 }
},
```

In the `properties` element for `config/connectionstrings`, each connection string is also defined as a name:value pair, with the specific format of `"<name>" : { "value": "...", "type": "..."}`. For the `type` element, possible values are `MySQL`, `SQLServer`, `SQLAzure`, and `Custom`.

#### TIP

For a definitive list of the connection string types, run the following command in Azure PowerShell:

```
[Enum]::GetNames("Microsoft.WindowsAzure.Commands.Utilities.Websites.Services.WebEntities.DatabaseType")
```

#### Source control

The source control settings are also defined as a nested resource. Azure Resource Manager uses this resource to configure continuous publishing (see caveat on `IsManualIntegration` later) and also to kick off the deployment of application code automatically during the processing of the JSON file.

```
{
 "apiVersion": "2015-04-01",
 "name": "web",
 "type": "sourcecontrols",
 "dependsOn": [
 "[resourceId('Microsoft.Web/Sites', variables('apiSiteName'))]",
 "[resourceId('Microsoft.Web/Sites/config', variables('apiSiteName'), 'appsettings')]",
 "[resourceId('Microsoft.Web/Sites/config', variables('apiSiteName'), 'connectionstrings')"
],
 "properties": {
 "RepoUrl": "[parameters('repoUrl')]",
 "branch": "[parameters('branch')]",
 "IsManualIntegration": true
 }
},
```

`RepoUrl` and `branch` should be pretty intuitive and should point to the Git repository and the name of the branch to publish from. Again, these are defined by input parameters.

Note in the `dependson` element that, in addition to the app resource itself, `sourcecontrols/web` also depends on `config/appsettings` and `config/connectionstrings`. This is because once `sourcecontrols/web` is configured, the Azure deployment process will automatically attempt to deploy, build, and start the application code. Therefore, inserting this dependency helps you make sure that the application has access to the required app settings and connection strings before the application code is run.

#### NOTE

Note also that `IsManualIntegration` is set to `true`. This property is necessary in this tutorial because you do not actually own the GitHub repository, and thus cannot actually grant permission to Azure to configure continuous publishing from [ToDoApp](#) (i.e. push automatic repository updates to Azure). You can use the default value `false` for the specified repository only if you have configured the owner's GitHub credentials in the [Azure portal](#) before. In other words, if you have set up source control to GitHub or BitBucket for any app in the [Azure Portal](#) previously, using your user credentials, then Azure will remember the credentials and use them whenever you deploy any app from GitHub or BitBucket in the future. However, if you haven't done this already, deployment of the JSON template will fail when Azure Resource Manager tries to configure the app's source control settings because it cannot log into GitHub or BitBucket with the repository owner's credentials.

## Compare the JSON template with deployed resource group

Here, you can go through all the app's blades in the [Azure Portal](#), but there's another tool that's just as useful, if not more. Go to the [Azure Resource Explorer](#) preview tool, which gives you a JSON representation of all the resource groups in your subscriptions, as they actually exist in the Azure backend. You can also see how the resource group's JSON hierarchy in Azure corresponds with the hierarchy in the template file that's used to create it.

For example, when I go to the [Azure Resource Explorer](#) tool and expand the nodes in the explorer, I can see the resource group and the root-level resources that are collected under their respective resource types.



If you drill down to an app, you should be able to see app configuration details similar to the below screenshot:

The screenshot shows the Azure Resource Explorer (Preview) interface. On the left, there's a tree view of resources under the 'ToDoApp9dd9' resource group. The 'appsettings' node is selected and highlighted in blue. The right pane shows the JSON content for the 'appsettings' resource, with a 'POST' button and an 'Edit' button above the JSON code.

```

1 - {
2 "id": "/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/sites/ToDoApp9dd9/config/appsettings",
3 "name": "appsettings",
4 "type": "Microsoft.Web/sites/config",
5 "kind": null,
6 "location": "Brazil South",
7 "tags": {
8 "hidden-related:/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/serverfarms/ToDoApp9dd9": "Resource"
9 },
10 "plan": null,
11 "properties": [
12 {
13 "name": "PROJECT",
14 "value": "src\\MultiChannelToDo\\MultiChannelToDo.csproj"
15 },
16 {
17 "name": "clientUrl",
18 "value": "http://ToDoApp9dd9.azurewebsites.net"
19 }
20]
21
22
23
24

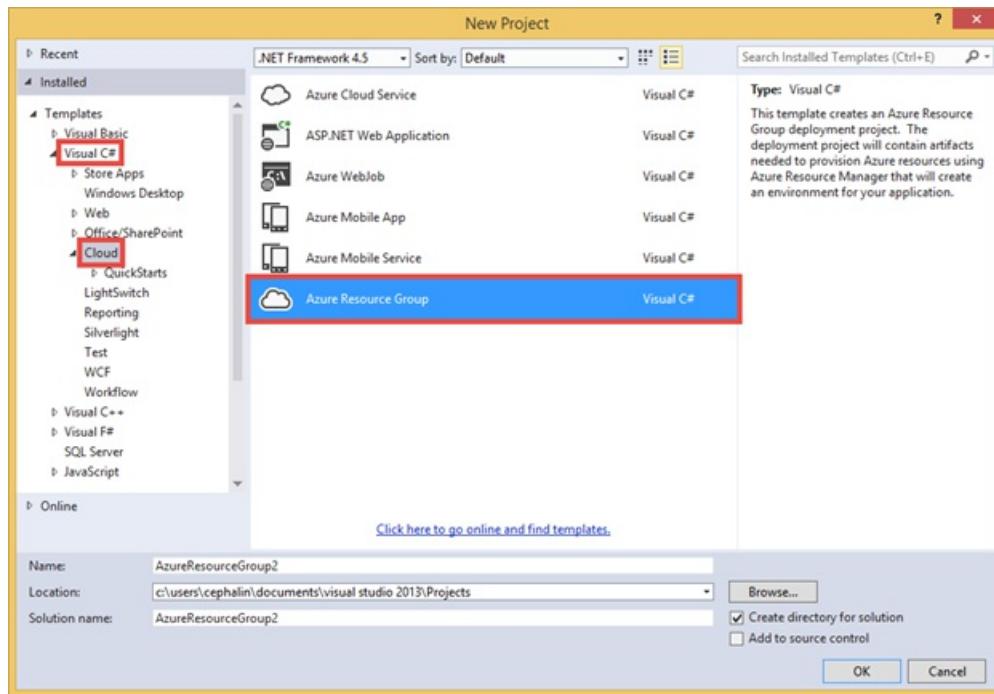
```

Again, the nested resources should have a hierarchy very similar to those in your JSON template file, and you should see the app settings, connection strings, etc., properly reflected in the JSON pane. The absence of settings here may indicate an issue with your JSON file and can help you troubleshoot your JSON template file.

## Deploy the resource group template yourself

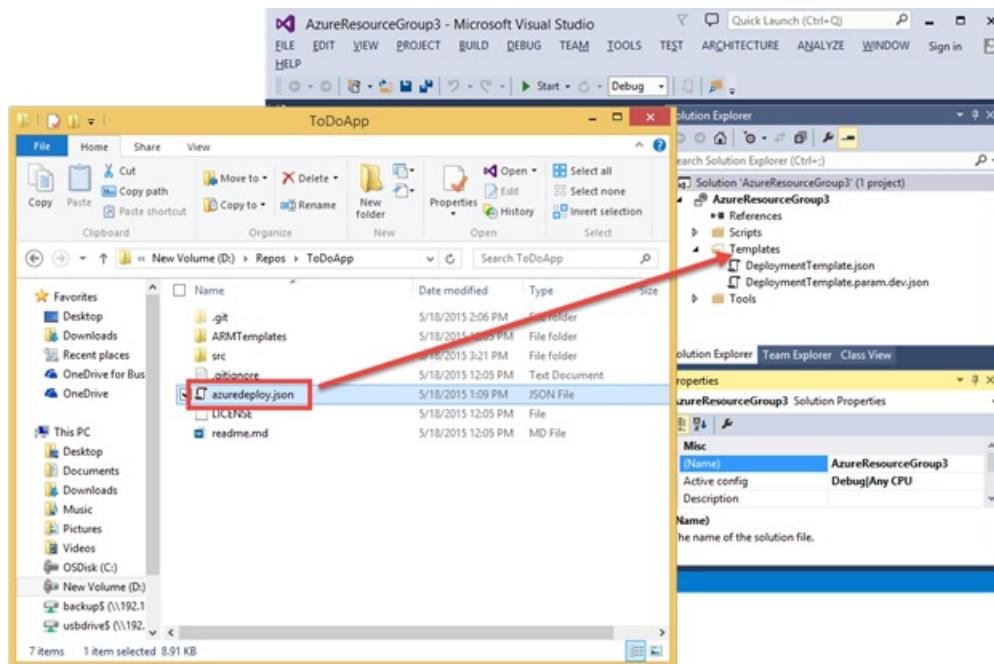
The **Deploy to Azure** button is great, but it allows you to deploy the resource group template in `azuredeploy.json` only if you have already pushed `azuredeploy.json` to GitHub. The Azure .NET SDK also provides the tools for you to deploy any JSON template file directly from your local machine. To do this, follow the steps below:

1. In Visual Studio, click **File > New > Project**.
2. Click **Visual C# > Cloud > Azure Resource Group**, then click **OK**.



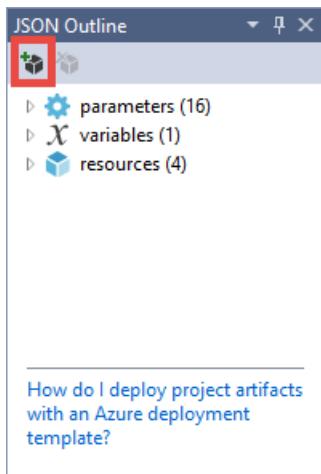
3. In Select Azure Template, select Blank Template and click OK.

4. Drag azuredploy.json into the Template folder of your new project.

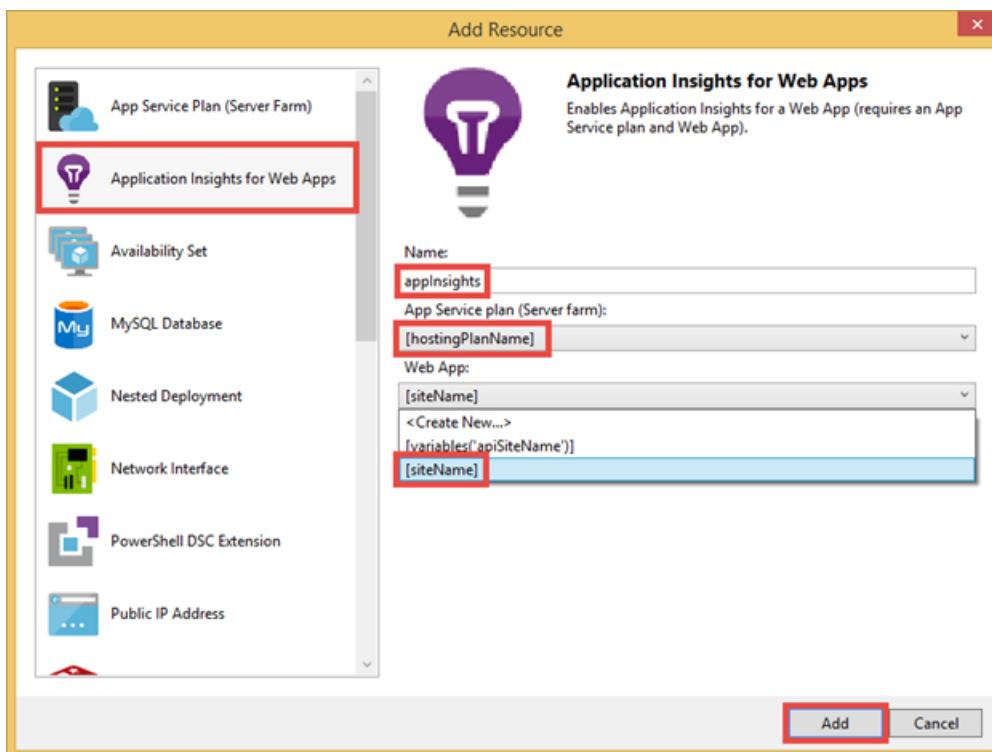


5. From Solution Explorer, open the copied azuredploy.json.

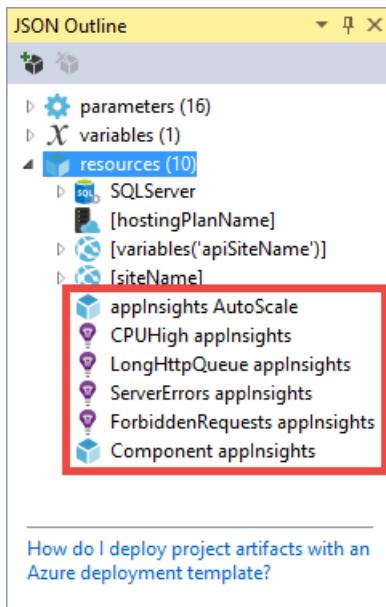
6. Just for the sake of the demonstration, let's add some standard Application Insight resources to our JSON file, by clicking Add Resource. If you're just interested in deploying the JSON file, skip to the deploy steps.



7. Select **Application Insights for Web Apps**, then make sure an existing App Service plan and app is selected, and then click **Add**.



You'll now be able to see several new resources that, depending on the resource and what it does, have dependencies on either the App Service plan or the app. These resources are not enabled by their existing definition and you're going to change that.



8. In the JSON Outline, click **applnights AutoScale** to highlight its JSON code. This is the scaling setting for your App Service plan.

9. In the highlighted JSON code, locate the `location` and `enabled` properties and set them as shown below.

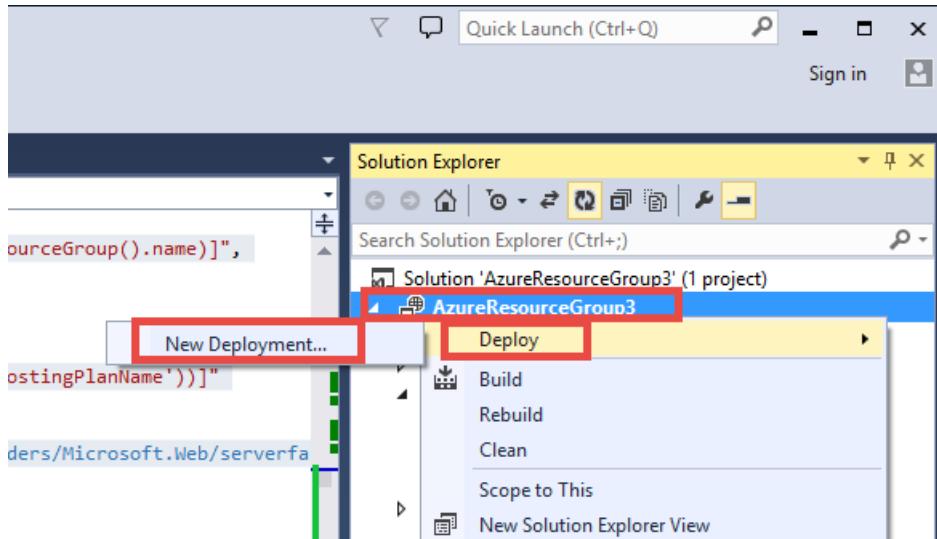
```
{
 "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
 "type": "Microsoft.Insights/autoscalesettings",
 "location": "[parameters('siteLocation')]",
 "apiVersion": "2014-04-01",
 "dependsOn": [...],
 "tags": [...],
 "properties": {
 "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name"
 "profiles": [...],
 "enabled": true,
 "targetResourceUri": "[concat(resourceGroup().id, '/providers/Microsoft.W
 }
},
}
```

10. In the JSON Outline, click **CPUHigh applnights** to highlight its JSON code. This is an alert.

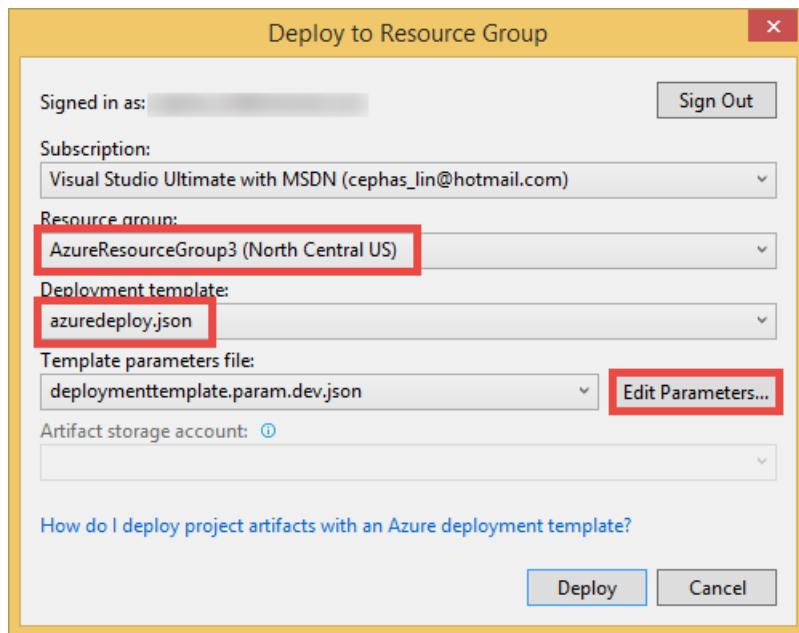
11. Locate the `location` and `isEnabled` properties and set them as shown below. Do the same for the other three alerts (purple bulbs).

```
{
 "name": "[concat('CPUHigh ', parameters('hostingPlanName'))]",
 "type": "Microsoft.Insights/alertrules",
 "location": "[parameters('siteLocation')]",
 "apiVersion": "2014-04-01",
 "dependsOn": [...],
 "tags": [...],
 "properties": {
 "name": "[concat('CPUHigh ', parameters('hostingPlanName'))]",
 "description": "[concat('The average CPU is high across all the instances')]",
 "isEnabled": true,
 "condition": [...],
 "action": [...]
 }
},
```

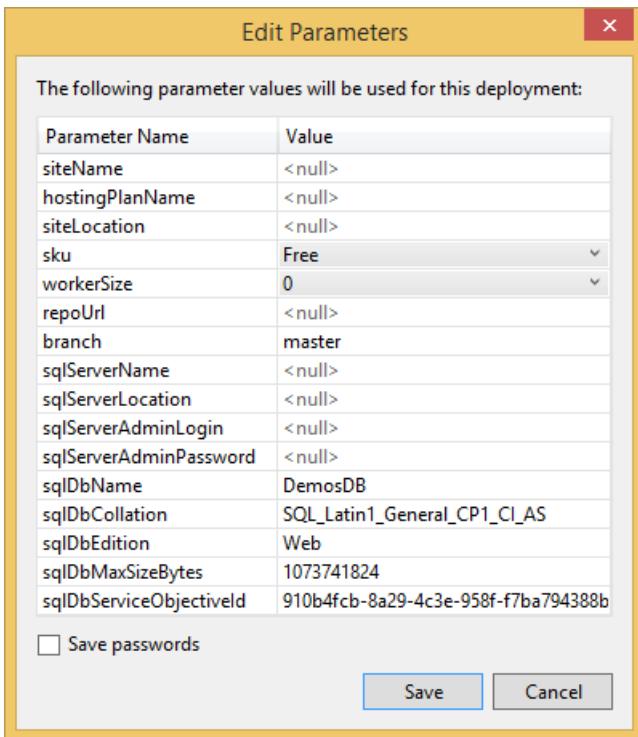
12. You're now ready to deploy. Right-click the project and select **Deploy > New Deployment**.



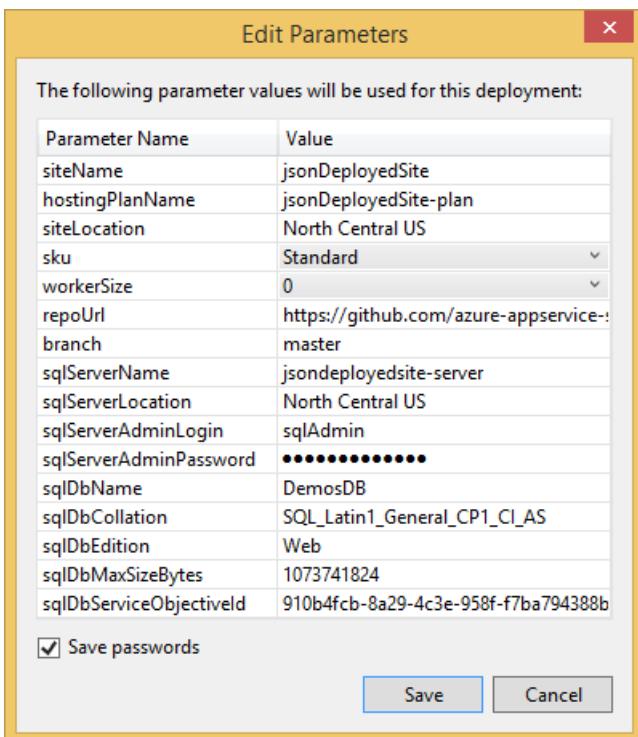
13. Log into your Azure account if you haven't already done so.
14. Select an existing resource group in your subscription or create a new one, select `azuredeploy.json`, and then click **Edit Parameters**.



You'll now be able to edit all the parameters defined in the template file in a nice table. Parameters that define defaults will already have their default values, and parameters that define a list of allowed values will be shown as dropdowns.



15. Fill in all the empty parameters, and use the [GitHub repo address](#) for **ToDoApp** in **repoUrl**. Then, click **Save**.



#### NOTE

Autoscaling is a feature offered in **Standard** tier or higher, and plan-level alerts are features offered in **Basic** tier or higher, you'll need to set the **sku** parameter to **Standard** or **Premium** in order to see all your new App Insights resources light up.

16. Click **Deploy**. If you selected **Save passwords**, the password will be saved in the parameter file **in plain text**. Otherwise, you'll be asked to input the database password during the deployment process.

That's it! Now you just need to go to the [Azure Portal](#) and the [Azure Resource Explorer](#) tool to see the new alerts and autoscale settings added to your JSON deployed application.

Your steps in this section mainly accomplished the following:

1. Prepared the template file
2. Created a parameter file to go with the template file
3. Deployed the template file with the parameter file

The last step is easily done by a PowerShell cmdlet. To see what Visual Studio did when it deployed your application, open Scripts\Deploy-AzureResourceGroup.ps1. There's a lot of code there, but I'm just going to highlight all the pertinent code you need to deploy the template file with the parameter file.

```
Set-StrictMode -Version 3
Import-Module Azure -ErrorAction SilentlyContinue

try {
 $AzureToolsUserAgentString = New-Object -TypeName System.Net.Http.Headers.ProductInfoHeaderValue -ArgumentList "AzCopy", "1.0"
 $headers = [System.Collections.Generic.Dictionary[[String], [String]]]::new()
 $headers.Add("User-Agent", $AzureToolsUserAgentString)
 $client = [System.Net.Http.HttpClient]::new()
 $client.DefaultRequestHeaders = $headers
}

Create or update the resource group using the specified template file and template parameters file
Switch-AzureMode AzureResourceManager
New-AzureResourceGroup -Name $ResourceGroupName `
 -Location $ResourceGroupLocation `
 -TemplateFile $TemplateFile `
 -TemplateParameterFile $TemplateParametersFile `
 @OptionalParameters `
 -Force -Verbose
```

The last cmdlet, `New-AzureResourceGroup`, is the one that actually performs the action. All this should demonstrate to you that, with the help of tooling, it is relatively straightforward to deploy your cloud application predictably. Every time you run the cmdlet on the same template with the same parameter file, you're going to get the same result.

## Summary

In DevOps, repeatability and predictability are keys to any successful deployment of a high-scale application composed of microservices. In this tutorial, you have deployed a two-microservice application to Azure as a single resource group using the Azure Resource Manager template. Hopefully, it has given you the knowledge you need in order to start converting your application in Azure into a template and can provision and deploy it predictably.

## More resources

- [Azure Resource Manager Template Language](#)
- [Authoring Azure Resource Manager templates](#)
- [Azure Resource Manager Template Functions](#)
- [Deploy an application with Azure Resource Manager template](#)
- [Using Azure PowerShell with Azure Resource Manager](#)
- [Troubleshooting Resource Group Deployments in Azure](#)

## Next steps

To learn about the JSON syntax and properties for resource types deployed in this article, see:

- [Microsoft.Sql/servers](#)
- [Microsoft.Sql/servers/databases](#)
- [Microsoft.Sql/servers/firewallRules](#)
- [Microsoft.Web/serverfarms](#)
- [Microsoft.Web/sites](#)
- [Microsoft.Web/sites/slots](#)

- Microsoft.Insights/autoscalesettings

# Configure deployment credentials for Azure App Service

11/2/2021 • 5 minutes to read • [Edit Online](#)

To secure app deployment from a local computer, [Azure App Service](#) supports two types of credentials for [local Git deployment](#) and [FTP/S deployment](#). These credentials are not the same as your Azure subscription credentials.

- **User-level credentials:** one set of credentials for the entire Azure account. It can be used to deploy to App Service for any app, in any subscription, that the Azure account has permission to access. It's the default set that's surfaced in the portal GUI (such as the [Overview](#) and [Properties](#) of the app's [resource page](#)). When a user is granted app access via Role-Based Access Control (RBAC) or coadmin permissions, that user can use their own user-level credentials until the access is revoked. Do not share these credentials with other Azure users.
- **App-level credentials:** one set of credentials for each app. It can be used to deploy to that app only. The credentials for each app are generated automatically at app creation. They can't be configured manually, but can be reset anytime. For a user to be granted access to app-level credentials via (RBAC), that user must be contributor or higher on the app (including Website Contributor built-in role). Readers are not allowed to publish, and can't access those credentials.

## NOTE

The [Development Center \(Classic\)](#) page in the Azure portal, which is the old deployment experience, will be deprecated in March, 2021. This change will not affect any existing deployment settings in your app, and you can continue to manage app deployment in the [Deployment Center](#) page.

## Configure user-scope credentials

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Azure portal](#)

Run the `az webapp deployment user set` command. Replace <username> and <password> with a deployment user username and password.

- The username must be unique within Azure, and for local Git pushes, must not contain the '@' symbol.
- The password must be at least eight characters long, with two of the following three elements: letters, numbers, and symbols.

```
az webapp deployment user set --user-name <username> --password <password>
```

The JSON output shows the password as `null`.

## Use user-scope credentials with FTP/FTPS

Authenticating to an FTP/FTPS endpoint using user-scope credentials requires a username in the following format: `<app-name>\<user-name>`

Since user-scope credentials are linked to the user and not a specific resource, the username must be in this format to direct the sign-in action to the right app endpoint.

## Get application-scope credentials

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Azure portal](#)

Get the application-scope credentials using the `az webapp deployment list-publishing-profiles` command. For example:

```
az webapp deployment list-publishing-profiles --resource-group <group-name> --name <app-name>
```

For [local Git deployment](#), you can also use the `az webapp deployment list-publishing-credentials` command to get a Git remote URI for your app, with the application-scope credentials already embedded. For example:

```
az webapp deployment list-publishing-credentials --resource-group <group-name> --name <app-name> --query scmUri
```

## Reset application-scope credentials

- [Azure CLI](#)
- [Azure PowerShell](#)
- [Azure portal](#)

Reset the application-scope credentials using the `az resource invoke-action` command:

```
az resource invoke-action --action newpassword --resource-group <group-name> --name <app-name> --resource-type Microsoft.Web/sites
```

## Disable basic authentication

Some organizations need to meet security requirements and would rather disable access via FTP or WebDeploy. This way, the organization's members can only access its App Services through APIs that are controlled by Azure Active Directory (Azure AD).

### FTP

To disable FTP access to the site, run the following CLI command. Replace the placeholders with your resource group and site name.

```
az resource update --resource-group <resource-group> --name ftp --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<site-name> --set properties.allow=false
```

To confirm that FTP access is blocked, you can try to authenticate using an FTP client such as FileZilla. To retrieve the publishing credentials, go to the overview blade of your site and click Download Publish Profile. Use the file's FTP hostname, username, and password to authenticate, and you will get a 401 error response, indicating that you are not authorized.

### WebDeploy and SCM

To disable basic auth access to the WebDeploy port and SCM site, run the following CLI command. Replace the

placeholders with your resource group and site name.

```
az resource update --resource-group <resource-group> --name scm --namespace Microsoft.Web --resource-type basicPublishingCredentialsPolicies --parent sites/<site-name> --set properties.allow=false
```

To confirm that the publish profile credentials are blocked on WebDeploy, try [publishing a web app using Visual Studio 2019](#).

### Disable access to the API

The API in the previous section is backed Azure role-based access control (Azure RBAC), which means you can [create a custom role](#) and assign lower-privileged users to the role so they cannot enable basic auth on any sites. To configure the custom role, [follow these instructions](#).

You can also use [Azure Monitor](#) to audit any successful authentication requests and use [Azure Policy](#) to enforce this configuration for all sites in your subscription.

## Next steps

Find out how to use these credentials to deploy your app from [local Git](#) or using [FTP/S](#).

# Set up staging environments in Azure App Service

11/2/2021 • 18 minutes to read • [Edit Online](#)

When you deploy your web app, web app on Linux, mobile back end, or API app to [Azure App Service](#), you can use a separate deployment slot instead of the default production slot when you're running in the **Standard**, **Premium**, or **Isolated** App Service plan tier. 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.

Deploying your application to a non-production slot has the following benefits:

- You can validate app changes in a staging deployment slot before swapping it with the production slot.
- Deploying an app to a slot first and swapping it into production makes sure that all instances of the slot are warmed up before being swapped into production. This eliminates downtime when you deploy your app. The traffic redirection is seamless, and no requests are dropped because of swap operations. You can automate this entire workflow by configuring [auto swap](#) when pre-swap validation isn't needed.
- After a swap, the slot with previously staged app now has the previous production app. If the changes swapped into the production slot aren't as you expect, you can perform the same swap immediately to get your "last known good site" back.

Each App Service plan tier supports a different number of deployment slots. There's no additional charge for using deployment slots. To find out the number of slots your app's tier supports, see [App Service limits](#).

To scale your app to a different tier, make sure that the target tier supports the number of slots your app already uses. For example, if your app has more than five slots, you can't scale it down to the **Standard** tier, because the **Standard** tier supports only five deployment slots.

## Add a slot

The app must be running in the **Standard**, **Premium**, or **Isolated** tier in order for you to enable multiple deployment slots.

1. in the [Azure portal](#), search for and select **App Services** and select your app.

The screenshot shows the Microsoft Azure portal's search interface. In the search bar at the top, the text 'app services' is typed. Below the search bar, a list of results titled 'Services' is displayed, with a total of 'All 60 results'. The first item in the list, 'App Services', is highlighted with a red box. Other items listed include 'Function App', 'Service Health', 'App Service Certificates', 'App Service Domains', 'App Service Environments', 'App Service plans', 'Lab Services', 'Media services', 'Bot Services', and 'Resources' (which shows 'No results were found'). On the left side of the portal, there are sections for 'Azure services' (with 'Create a resource' and 'Resource groups' buttons), 'Recent resources' (listing 'myFirstAzureWebA', 'WebApplicationAS', and 'cs4316e81020662x'), and 'Navigate' (with 'Subscriptions' button). The top right corner shows the user's email 'Admin@contoso.com' and the account name 'CONTOSO'.

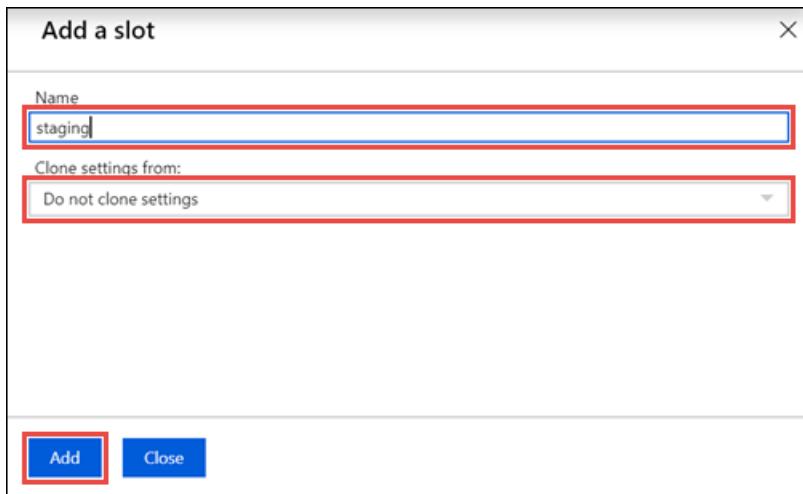
2. In the left pane, select Deployment slots > Add Slot.

The screenshot shows the 'Deployment slots' page for the app 'my-demo-app'. The left sidebar has a 'Deployment slots' item highlighted with a red box. The main area displays a message: 'You haven't added any deployment slots. Click here to get started.' Below this is a 'Deployment Slots' section with a sub-section titled 'Deployment Slots'. It contains a brief description: 'Deployment slots are live apps with their own hostnames. App content and configurations elements can be swapped between two deployment slots, including the production slot.' A table lists one deployment slot: 'my-demo-app' (NAME), 'Running' (STATUS), 'myAppServicePlan' (APP SERVICE PLAN), and '100' (TRAFFIC %). At the top of the page, there are buttons for 'Save', 'Discard', 'Swap', and 'Refresh', with the '+ Add Slot' button highlighted by a red box.

#### NOTE

If the app isn't already in the **Standard**, **Premium**, or **Isolated** tier, you receive a message that indicates the supported tiers for enabling staged publishing. At this point, you have the option to select **Upgrade** and go to the **Scale** tab of your app before continuing.

3. In the **Add a slot** dialog box, give the slot a name, and select whether to clone an app configuration from another deployment slot. Select **Add** to continue.



You can clone a configuration from any existing slot. Settings that can be cloned include app settings, connection strings, language framework versions, web sockets, HTTP version, and platform bitness.

4. After the slot is added, select **Close** to close the dialog box. The new slot is now shown on the **Deployment slots** page. By default, **Traffic %** is set to 0 for the new slot, with all customer traffic routed to the production slot.
5. Select the new deployment slot to open that slot's resource page.

The screenshot shows the 'Deployment slots' page for the 'my-demo-app' app service. The left sidebar has a 'Deployment slots' section selected. The main area displays a table of deployment slots:

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app <span style="background-color: green; color: white;">PRODUCTION</span>	Running	myAppServicePlan	100
my-demo-app-staging	Running	myAppServicePlan	0

The staging slot has a management page just like any other App Service app. You can change the slot's configuration. To remind you that you're viewing the deployment slot, the app name is shown as <app-name>/<slot-name>, and the app type is **App Service (Slot)**. You can also see the slot as a separate app in your resource group, with the same designations.

6. Select the app URL on the slot's resource page. The deployment slot has its own host name and is also a live app. To limit public access to the deployment slot, see [Azure App Service IP restrictions](#).

The new deployment slot has no content, even if you clone the settings from a different slot. For example, you can [publish to this slot with Git](#). You can deploy to the slot from a different repository branch or a different repository.

The slot's URL will be of the format `http://sitename-slotname.azurewebsites.net`. To keep the URL length within necessary DNS limits, the site name will be truncated at 40 characters, the slot name will be truncated at 19 characters, and an additional 4 random characters will be appended to ensure the resulting domain name is unique.

# What happens during a swap

## Swap operation steps

When you swap two slots (usually from a staging slot into the production slot), App Service does the following to ensure that the target slot doesn't experience downtime:

1. Apply the following settings from the target slot (for example, the production slot) to all instances of the source slot:
  - [Slot-specific](#) app settings and connection strings, if applicable.
  - [Continuous deployment](#) settings, if enabled.
  - [App Service authentication](#) settings, if enabled.Any of these cases trigger all instances in the source slot to restart. During [swap with preview](#), this marks the end of the first phase. The swap operation is paused, and you can validate that the source slot works correctly with the target slot's settings.
2. Wait for every instance in the source slot to complete its restart. If any instance fails to restart, the swap operation reverts all changes to the source slot and stops the operation.
3. If [local cache](#) is enabled, trigger local cache initialization by making an HTTP request to the application root ("/") on each instance of the source slot. Wait until each instance returns any HTTP response. Local cache initialization causes another restart on each instance.
4. If [auto swap](#) is enabled with [custom warm-up](#), trigger [Application Initiation](#) by making an HTTP request to the application root ("/") on each instance of the source slot.

If `applicationInitialization` isn't specified, trigger an HTTP request to the application root of the source slot on each instance.

If an instance returns any HTTP response, it's considered to be warmed up.

5. If all instances on the source slot are warmed up successfully, swap the two slots by switching the routing rules for the two slots. After this step, the target slot (for example, the production slot) has the app that's previously warmed up in the source slot.
6. Now that the source slot has the pre-swap app previously in the target slot, perform the same operation by applying all settings and restarting the instances.

At any point of the swap operation, all work of initializing the swapped apps happens on the source slot. The target slot remains online while the source slot is being prepared and warmed up, regardless of where the swap succeeds or fails. To swap a staging slot with the production slot, make sure that the production slot is always the target slot. This way, the swap operation doesn't affect your production app.

### NOTE

The instances in your former production instances (those that will be swapped into staging after this swap operation) will be recycled quickly in the last step of the swap process. In case you have any long running operations in your application, they will be abandoned, when the workers recycle. This also applies to function apps. Therefore your application code should be written in a fault tolerant way.

## Which settings are swapped?

When you clone configuration from another deployment slot, the cloned configuration is editable. Some configuration elements follow the content across a swap (not slot specific), whereas other configuration elements stay in the same slot after a swap (slot specific). The following lists show the settings that change when you swap slots.

## Settings that are swapped:

- General settings, such as framework version, 32/64-bit, web sockets
- App settings (can be configured to stick to a slot)
- Connection strings (can be configured to stick to a slot)
- Handler mappings
- Public certificates
- WebJobs content
- Hybrid connections \*
- Service endpoints \*
- Azure Content Delivery Network \*
- Path mappings

Features marked with an asterisk (\*) are planned to be unswapped.

## Settings that aren't swapped:

- Publishing endpoints
- Custom domain names
- Non-public certificates and TLS/SSL settings
- Scale settings
- WebJobs schedulers
- IP restrictions
- Always On
- Diagnostic settings
- Cross-origin resource sharing (CORS)
- Virtual network integration
- Managed identities
- Settings that end with the suffix \_EXTENSION\_VERSION

### NOTE

To make aforementioned settings swappable, add the app setting

`WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SLOT_SETTINGS` in every slot of the app and set its value to `0` or `false`. These settings are either all swappable or not at all. You can't make just some settings swappable and not the others. Managed identities are never swapped and are not affected by this override app setting.

Certain app settings that apply to unswapped settings are also not swapped. For example, since diagnostic settings are not swapped, related app settings like `WEBSITE_HTTPLOGGING_RETENTION_DAYS` and `DIAGNOSTICS_AZUREBLOBRETENTIONDAYS` are also not swapped, even if they don't show up as slot settings.

To configure an app setting or connection string to stick to a specific slot (not swapped), go to the **Configuration** page for that slot. Add or edit a setting, and then select **deployment slot setting**. Selecting this check box tells App Service that the setting is not swappable.

Add/Edit application setting

Name	Debug
Value	false
<input type="checkbox"/> deployment slot setting	
<b>Update</b>	<b>Cancel</b>

## Swap two slots

You can swap deployment slots on your app's **Deployment slots** page and the **Overview** page. For technical details on the slot swap, see [What happens during swap](#).

### IMPORTANT

Before you swap an app from a deployment slot into production, make sure that production is your target slot and that all settings in the source slot are configured exactly as you want to have them in production.

To swap deployment slots:

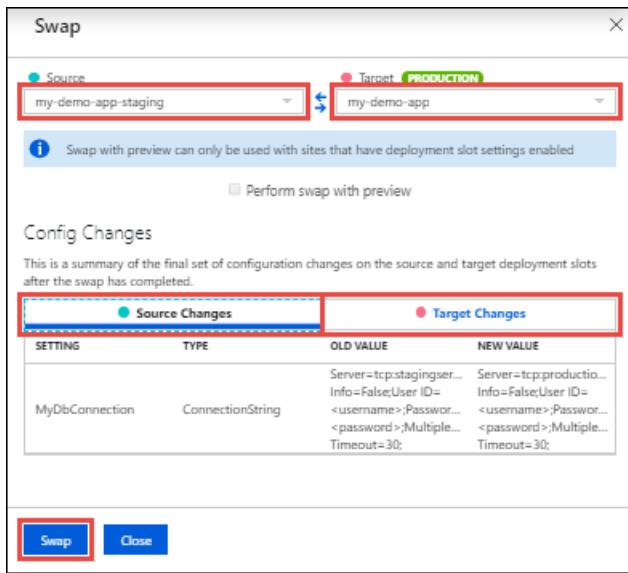
1. Go to your app's **Deployment slots** page and select **Swap**.

The screenshot shows the 'my-demo-app - Deployment slots' blade. On the left, a sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment, Quickstart, Deployment slots (which is selected and highlighted with a red box), Deployment Center, Settings, and Configuration. At the top right, there are Save, Discard, Add Slot, Swap (which is highlighted with a red box), and Refresh buttons. The main area is titled 'Deployment Slots' and contains the following text: 'Deployment slots are live apps with their own hostnames. App content and configurations elements can be swapped between two deployment slots, including the production slot.' Below this is a table with the following data:

NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app <b>PRODUCTION</b>	Running	myAppServicePlan	100
my-demo-app-staging	Running	myAppServicePlan	0

The **Swap** dialog box shows settings in the selected source and target slots that will be changed.

2. Select the desired **Source** and **Target** slots. Usually, the target is the production slot. Also, select the **Source Changes** and **Target Changes** tabs and verify that the configuration changes are expected. When you're finished, you can swap the slots immediately by selecting **Swap**.



To see how your target slot would run with the new settings before the swap actually happens, don't select **Swap**, but follow the instructions in [Swap with preview](#).

- When you're finished, close the dialog box by selecting **Close**.

If you have any problems, see [Troubleshoot swaps](#).

### Swap with preview (multi-phase swap)

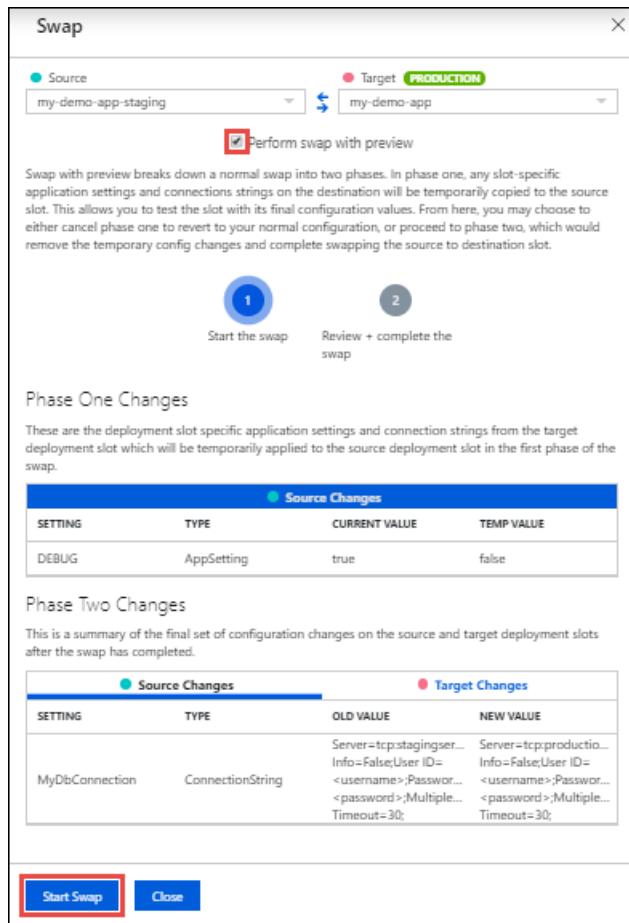
Before you swap into production as the target slot, validate that the app runs with the swapped settings. The source slot is also warmed up before the swap completion, which is desirable for mission-critical applications.

When you perform a swap with preview, App Service performs the same [swap operation](#) but pauses after the first step. You can then verify the result on the staging slot before completing the swap.

If you cancel the swap, App Service reapplies configuration elements to the source slot.

To swap with preview:

- Follow the steps in [Swap deployment slots](#) but select **Perform swap with preview**.



The dialog box shows you how the configuration in the source slot changes in phase 1, and how the source and target slot change in phase 2.

## 2. When you're ready to start the swap, select **Start Swap**.

When phase 1 finishes, you're notified in the dialog box. Preview the swap in the source slot by going to [https://<app\\_name>-<source-slot-name>.azurewebsites.net](https://<app_name>-<source-slot-name>.azurewebsites.net).

## 3. When you're ready to complete the pending swap, select **Complete Swap** in **Swap action** and select **Complete Swap**.

To cancel a pending swap, select **Cancel Swap** instead.

## 4. When you're finished, close the dialog box by selecting **Close**.

If you have any problems, see [Troubleshoot swaps](#).

To automate a multi-phase swap, see [Automate with PowerShell](#).

## Roll back a swap

If any errors occur in the target slot (for example, the production slot) after a slot swap, restore the slots to their pre-swap states by swapping the same two slots immediately.

## Configure auto swap

### NOTE

Auto swap isn't supported in web apps on Linux and Web App for Containers.

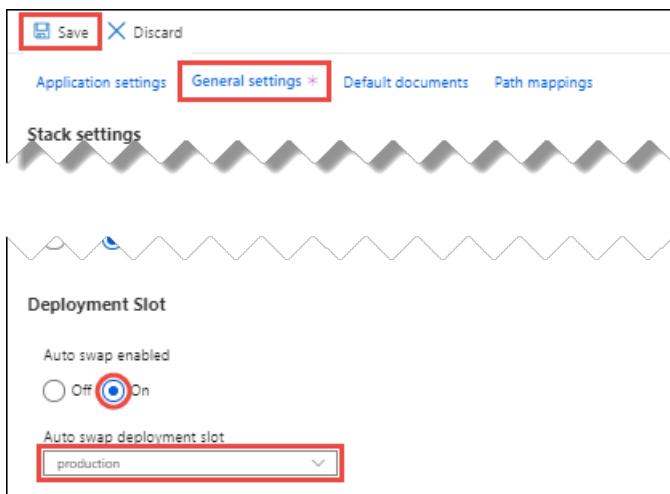
Auto swap streamlines Azure DevOps scenarios where you want to deploy your app continuously with zero cold starts and zero downtime for customers of the app. When auto swap is enabled from a slot into production, every time you push your code changes to that slot, App Service automatically [swaps the app into production](#) after it's warmed up in the source slot.

**NOTE**

Before you configure auto swap for the production slot, consider testing auto swap on a non-production target slot.

To configure auto swap:

1. Go to your app's resource page. Select **Deployment slots** > *<desired source slot>* > **Configuration** > **General settings**.
2. For **Auto swap enabled**, select **On**. Then select the desired target slot for **Auto swap deployment slot**, and select **Save** on the command bar.



3. Execute a code push to the source slot. Auto swap happens after a short time, and the update is reflected at your target slot's URL.

If you have any problems, see [Troubleshoot swaps](#).

## Specify custom warm-up

Some apps might require custom warm-up actions before the swap. The `applicationInitialization` configuration element in `web.config` lets you specify custom initialization actions. The [swap operation](#) waits for this custom warm-up to finish before swapping with the target slot. Here's a sample `web.config` fragment.

```
<system.webServer>
 <applicationInitialization>
 <add initializationPage="/" hostName="[app hostname]" />
 <add initializationPage="/Home/About" hostName="[app hostname]" />
 </applicationInitialization>
</system.webServer>
```

For more information on customizing the `applicationInitialization` element, see [Most common deployment slot swap failures and how to fix them](#).

You can also customize the warm-up behavior with one or both of the following [app settings](#):

- `WEBSITE_SWAP_WARMUP_PING_PATH`: The path to ping to warm up your site. Add this app setting by specifying a

custom path that begins with a slash as the value. An example is `/statuscheck`. The default value is `/`.

- `WEBSITE_SWAP_WARMUP_PING_STATUSES`: Valid HTTP response codes for the warm-up operation. Add this app setting with a comma-separated list of HTTP codes. An example is `200,202`. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.
- `WEBSITE_WARMUP_PATH`: A relative path on the site that should be pinged whenever the site restarts (not only during slot swaps). Example values include `/statuscheck` or the root path, `/`.

#### NOTE

The `<applicationInitialization>` configuration element is part of each app start-up, whereas the two warm-up behavior app settings apply only to slot swaps.

If you have any problems, see [Troubleshoot swaps](#).

## Monitor a swap

If the [swap operation](#) takes a long time to complete, you can get information on the swap operation in the [activity log](#).

On your app's resource page in the portal, in the left pane, select **Activity log**.

A swap operation appears in the log query as `Swap Web App Slots`. You can expand it and select one of the suboperations or errors to see the details.

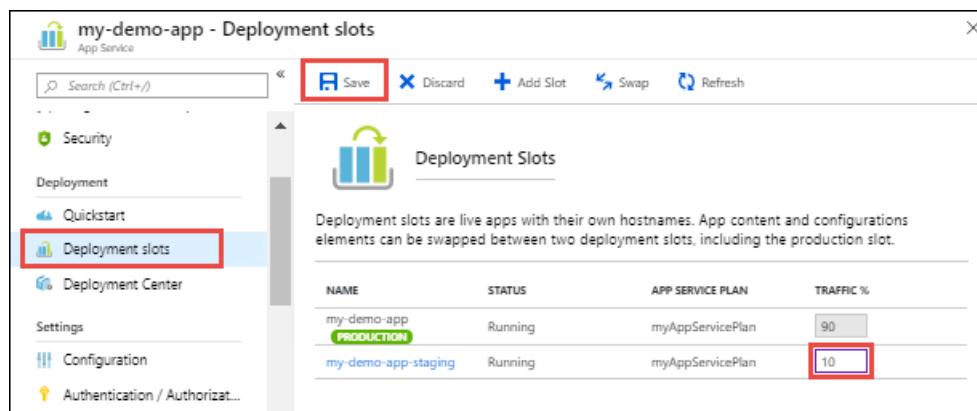
## Route traffic

By default, all client requests to the app's production URL (`http://<app_name>.azurewebsites.net`) are routed to the production slot. You can route a portion of the traffic to another slot. This feature is useful if you need user feedback for a new update, but you're not ready to release it to production.

### Route production traffic automatically

To route production traffic automatically:

1. Go to your app's resource page and select **Deployment slots**.
2. In the **Traffic %** column of the slot you want to route to, specify a percentage (between 0 and 100) to represent the amount of total traffic you want to route. Select **Save**.



NAME	STATUS	APP SERVICE PLAN	TRAFFIC %
my-demo-app	Running	myAppServicePlan	90
my-demo-app-staging	Running	myAppServicePlan	10

After the setting is saved, the specified percentage of clients is randomly routed to the non-production slot.

After a client is automatically routed to a specific slot, it's "pinned" to that slot for the life of that client session. On the client browser, you can see which slot your session is pinned to by looking at the `x-ms-routing-name` cookie in your HTTP headers. A request that's routed to the "staging" slot has the cookie

`x-ms-routing-name=staging`. A request that's routed to the production slot has the cookie `x-ms-routing-name=self`.

#### NOTE

You can also use the `az webapp traffic-routing set` command in the Azure CLI to set the routing percentages from CI/CD tools like GitHub Actions, DevOps pipelines, or other automation systems.

### Route production traffic manually

In addition to automatic traffic routing, App Service can route requests to a specific slot. This is useful when you want your users to be able to opt in to or opt out of your beta app. To route production traffic manually, you use the `x-ms-routing-name` query parameter.

To let users opt out of your beta app, for example, you can put this link on your webpage:

```
<a href="<webappname>.azurewebsites.net/?x-ms-routing-name=self">Go back to production app
```

The string `x-ms-routing-name=self` specifies the production slot. After the client browser accesses the link, it's redirected to the production slot. Every subsequent request has the `x-ms-routing-name=self` cookie that pins the session to the production slot.

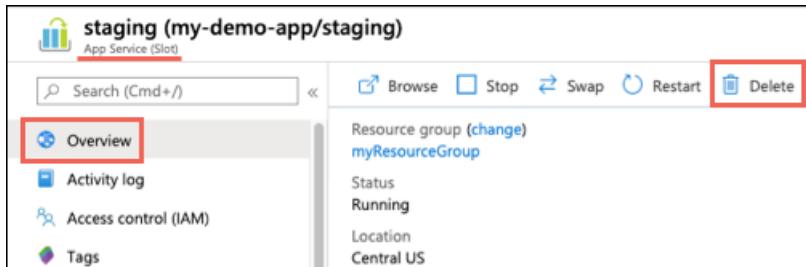
To let users opt in to your beta app, set the same query parameter to the name of the non-production slot. Here's an example:

```
<webappname>.azurewebsites.net/?x-ms-routing-name=staging
```

By default, new slots are given a routing rule of `0%`, shown in grey. When you explicitly set this value to `0%` (shown in black text), your users can access the staging slot manually by using the `x-ms-routing-name` query parameter. But they won't be routed to the slot automatically because the routing percentage is set to 0. This is an advanced scenario where you can "hide" your staging slot from the public while allowing internal teams to test changes on the slot.

## Delete a slot

Search for and select your app. Select **Deployment slots** > `<slot to delete>` > **Overview**. The app type is shown as **App Service (Slot)** to remind you that you're viewing a deployment slot. Select **Delete** on the command bar.



## Automate with PowerShell

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Azure PowerShell is a module that provides cmdlets to manage Azure through Windows PowerShell, including support for managing deployment slots in Azure App Service.

For information on installing and configuring Azure PowerShell, and on authenticating Azure PowerShell with your Azure subscription, see [How to install and configure Microsoft Azure PowerShell](#).

### Create a web app

```
New-AzWebApp -ResourceGroupName [resource group name] -Name [app name] -Location [location] -AppServicePlan [app service plan name]
```

### Create a slot

```
New-AzWebAppSlot -ResourceGroupName [resource group name] -Name [app name] -Slot [deployment slot name] -AppServicePlan [app service plan name]
```

### Initiate a swap with a preview (multi-phase swap), and apply destination slot configuration to the source slot

```
$ParametersObject = @{targetSlot = "[slot name - e.g. \"production\"]"}
Invoke-AzResourceAction -ResourceGroupName [resource group name] -ResourceType Microsoft.Web/sites/slots -ResourceName [app name]/[slot name] -Action applySlotConfig -Parameters $ParametersObject -ApiVersion 2015-07-01
```

### Cancel a pending swap (swap with review) and restore the source slot configuration

```
Invoke-AzResourceAction -ResourceGroupName [resource group name] -ResourceType Microsoft.Web/sites/slots -ResourceName [app name]/[slot name] -Action resetSlotConfig -ApiVersion 2015-07-01
```

### Swap deployment slots

```
$ParametersObject = @{targetSlot = "[slot name - e.g. \"production\"]"}
Invoke-AzResourceAction -ResourceGroupName [resource group name] -ResourceType Microsoft.Web/sites/slots -ResourceName [app name]/[slot name] -Action slotsswap -Parameters $ParametersObject -ApiVersion 2015-07-01
```

### Monitor swap events in the activity log

```
Get-AzLog -ResourceGroup [resource group name] -StartTime 2018-03-07 -Caller SlotSwapJobProcessor
```

### Delete a slot

```
Remove-AzResource -ResourceGroupName [resource group name] -ResourceType Microsoft.Web/sites/slots -Name [app name]/[slot name] -ApiVersion 2015-07-01
```

# Automate with Resource Manager templates

Azure Resource Manager templates are declarative JSON files used to automate the deployment and configuration of Azure resources. To swap slots by using Resource Manager templates, you will set two properties on the *Microsoft.Web/sites/slots* and *Microsoft.Web/sites* resources:

- `buildVersion`: this is a string property which represents the current version of the app deployed in the slot. For example: "v1", "1.0.0.1", or "2019-09-20T11:53:25.2887393-07:00".
- `targetBuildVersion`: this is a string property that specifies what `buildVersion` the slot should have. If the `targetBuildVersion` does not equal the current `buildVersion`, then this will trigger the swap operation by finding the slot which has the specified `buildVersion`.

## Example Resource Manager template

The following Resource Manager template will update the `buildVersion` of the staging slot and set the `targetBuildVersion` on the production slot. This will swap the two slots. The template assumes you already have a webapp created with a slot named "staging".

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "my_site_name": {
 "defaultValue": "SwapAPIDemo",
 "type": "String"
 },
 "sites_buildVersion": {
 "defaultValue": "v1",
 "type": "String"
 }
 },
 "resources": [
 {
 "type": "Microsoft.Web/sites/slots",
 "apiVersion": "2018-02-01",
 "name": "[concat(parameters('my_site_name'), '/staging')]",
 "location": "East US",
 "kind": "app",
 "properties": {
 "buildVersion": "[parameters('sites_buildVersion')]"
 }
 },
 {
 "type": "Microsoft.Web/sites",
 "apiVersion": "2018-02-01",
 "name": "[parameters('my_site_name')]",
 "location": "East US",
 "kind": "app",
 "dependsOn": [
 "[resourceId('Microsoft.Web/sites/slots', parameters('my_site_name'), 'staging')]"
],
 "properties": {
 "targetBuildVersion": "[parameters('sites_buildVersion')]"
 }
 }
]
}
```

This Resource Manager template is idempotent, meaning that it can be executed repeatedly and produce the same state of the slots. After the first execution, `targetBuildVersion` will match the current `buildVersion`, so a swap will not be triggered.

# Automate with the CLI

For Azure CLI commands for deployment slots, see [az webapp deployment slot](#).

## Troubleshoot swaps

If any error occurs during a [slot swap](#), it's logged in *D:\home\LogFiles\eventlog.xml*. It's also logged in the application-specific error log.

Here are some common swap errors:

- An HTTP request to the application root is timed. The swap operation waits for 90 seconds for each HTTP request, and retries up to 5 times. If all retries are timed out, the swap operation is stopped.
- Local cache initialization might fail when the app content exceeds the local disk quota specified for the local cache. For more information, see [Local cache overview](#).
- During [custom warm-up](#), the HTTP requests are made internally (without going through the external URL). They can fail with certain URL rewrite rules in *Web.config*. For example, rules for redirecting domain names or enforcing HTTPS can prevent warm-up requests from reaching the app code. To work around this issue, modify your rewrite rules by adding the following two conditions:

```
<conditions>
 <add input="{WARMUP_REQUEST}" pattern="1" negate="true" />
 <add input="{REMOTE_ADDR}" pattern="^100?\.+" negate="true" />
 ...
</conditions>
```

- Without a custom warm-up, the URL rewrite rules can still block HTTP requests. To work around this issue, modify your rewrite rules by adding the following condition:

```
<conditions>
 <add input="{REMOTE_ADDR}" pattern="^100?\.+" negate="true" />
 ...
</conditions>
```

- After slot swaps, the app may experience unexpected restarts. This is because after a swap, the hostname binding configuration goes out of sync, which by itself doesn't cause restarts. However, certain underlying storage events (such as storage volume failovers) may detect these discrepancies and force all worker processes to restart. To minimize these types of restarts, set the [WEBSITE\\_ADD\\_SITENAME\\_BINDINGS\\_IN\\_APPHOST\\_CONFIG=1](#) app setting on *all slots*. However, this app setting does *not* work with Windows Communication Foundation (WCF) apps.

## Next steps

[Block access to non-production slots](#)

# Guidance on deploying web apps by using Azure Resource Manager templates

11/2/2021 • 3 minutes to read • [Edit Online](#)

This article provides recommendations for creating Azure Resource Manager templates to deploy Azure App Service solutions. These recommendations can help you avoid common problems.

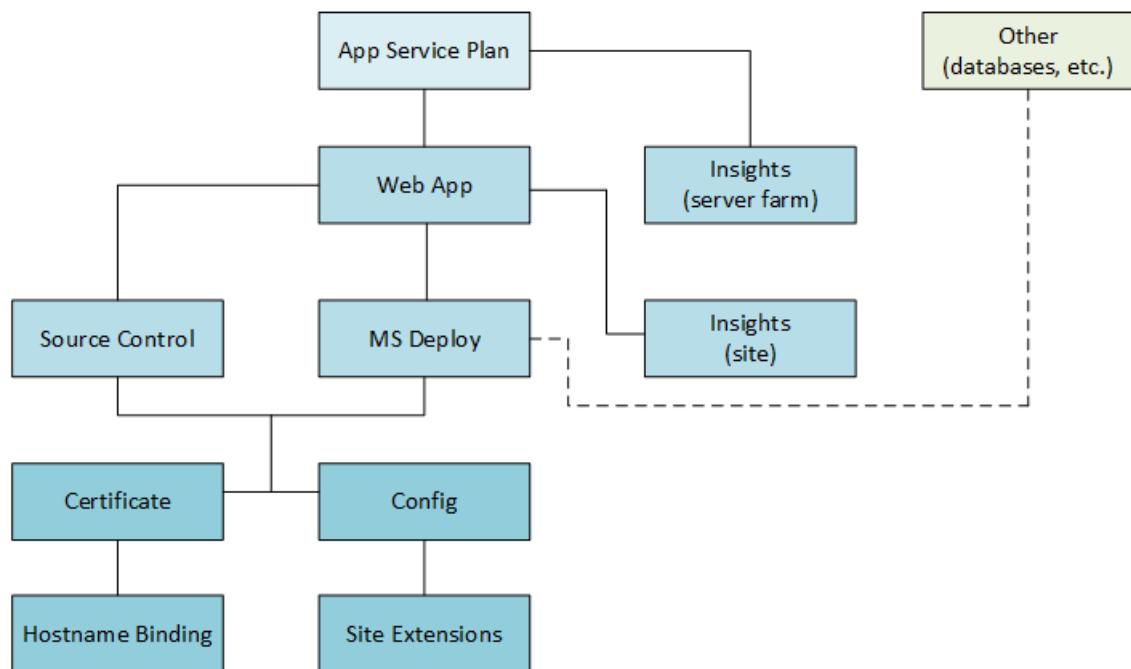
## Define dependencies

Defining dependencies for web apps requires an understanding of how the resources within a web app interact. If you specify dependencies in an incorrect order, you might cause deployment errors or create a race condition that stalls the deployment.

### WARNING

If you include an MSDeploy site extension in your template, you must set any configuration resources as dependent on the MSDeploy resource. Configuration changes cause the site to restart asynchronously. By making the configuration resources dependent on MSDeploy, you ensure that MSDeploy finishes before the site restarts. Without these dependencies, the site might restart during the deployment process of MSDeploy. For an example template, see [WordPress Template with Web Deploy Dependency](#).

The following image shows the dependency order for various App Service resources:



You deploy resources in the following order:

### Tier 1

- App Service plan.
- Any other related resources, like databases or storage accounts.

### Tier 2

- Web app--depends on the App Service plan.

- Azure Application Insights instance that targets the server farm--depends on the App Service plan.

#### Tier 3

- Source control--depends on the web app.
- MSDeploy site extension--depends on the web app.
- Azure Application Insights instance that targets the web app--depends on the web app.

#### Tier 4

- App Service certificate--depends on source control or MSDeploy if either is present. Otherwise, it depends on the web app.
- Configuration settings (connection strings, web.config values, app settings)--depends on source control or MSDeploy if either is present. Otherwise, it depends on the web app.

#### Tier 5

- Host name bindings--depends on the certificate if present. Otherwise, it depends on a higher-level resource.
- Site extensions--depends on configuration settings if present. Otherwise, it depends on a higher-level resource.

Typically, your solution includes only some of these resources and tiers. For missing tiers, map lower resources to the next-higher tier.

The following example shows part of a template. The value of the connection string configuration depends on the MSDeploy extension. The MSDeploy extension depends on the web app and database.

```
{
 "name": "[parameters('appName')]",
 "type": "Microsoft.Web/Sites",
 ...
 "resources": [
 {
 "name": "MSDeploy",
 "type": "Extensions",
 "dependsOn": [
 "[concat('Microsoft.Web/Sites/', parameters('appName'))]",
 "[concat('Microsoft.Sql/servers/', parameters('dbServerName'), '/databases/',
 parameters('dbName'))]"
],
 ...
 },
 {
 "name": "connectionstrings",
 "type": "config",
 "dependsOn": [
 "[concat('Microsoft.Web/Sites/', parameters('appName'), '/Extensions/MSDeploy')]"
],
 ...
 }
]
}
```

For a ready-to-run sample that uses the code above, see [Template: Build a simple Umbraco Web App](#).

## Find information about MSDeploy errors

If your Resource Manager template uses MSDeploy, the deployment error messages can be difficult to understand. To get more information after a failed deployment, try the following steps:

1. Go to the site's [Kudu console](#).

2. Browse to the folder at D:\home\LogFiles\SiteExtensions\MSDeploy.
3. Look for the appManagerStatus.xml and appManagerLog.xml files. The first file logs the status. The second file logs information about the error. If the error isn't clear to you, you can include it when you're asking for help on the [forum](#).

## Choose a unique web app name

The name for your web app must be globally unique. You can use a naming convention that's likely to be unique, or you can use the [uniqueString function](#) to assist with generating a unique name.

```
{
 "apiVersion": "2016-08-01",
 "name": "[concat(parameters('siteNamePrefix'), uniqueString(resourceGroup().id))]",
 "type": "Microsoft.Web/sites",
 ...
}
```

## Deploy web app certificate from Key Vault

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

If your template includes a [Microsoft.Web/certificates](#) resource for TLS/SSL binding, and the certificate is stored in a Key Vault, you must make sure the App Service identity can access the certificate.

In global Azure, the App Service service principal has the ID of **abfa0a7c-a6b6-4736-8310-5855508787cd**. To grant access to Key Vault for the App Service service principal, use:

```
Set-AzKeyVaultAccessPolicy `
 -VaultName KEY_VAULT_NAME `
 -ServicePrincipalName abfa0a7c-a6b6-4736-8310-5855508787cd `
 -PermissionsToSecrets get `
 -PermissionsToCertificates get
```

In Azure Government, the App Service service principal has the ID of **6a02c803-daf3-4136-b4c3-5a6f318b4714**. Use that ID in the preceding example.

In your Key Vault, select **Certificates** and **Generate/Import** to upload the certificate.

The screenshot shows the 'Certificates' blade in the Azure Key Vault interface. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Keys, Secrets, and Certificates. The 'Certificates' option is selected and highlighted with a red box. At the top right, there are two buttons: 'Generate/Import' (also highlighted with a red box) and 'Refresh'. The main area is titled 'NAME' and displays the message 'There are no certificates available.'

In your template, provide the name of the certificate for the `keyVaultSecretName`.

For an example template, see [Deploy a Web App certificate from Key Vault secret and use it for creating SSL binding](#).

## Next steps

- For a tutorial on deploying web apps with a template, see [Provision and deploy microservices predictably in Azure](#).
- To learn about JSON syntax and properties for resource types in templates, see [Azure Resource Manager template reference](#).

# Buy a custom domain name for Azure App Service

11/2/2021 • 9 minutes to read • [Edit Online](#)

App Service domains are custom domains that are managed directly in Azure. They make it easy to manage custom domains for [Azure App Service](#). This tutorial shows you how to buy an App Service domain and assign DNS names to Azure App Service.

For Azure VM or Azure Storage, see [Assign App Service domain to Azure VM or Azure Storage](#). For Cloud Services, see [Configuring a custom domain name for an Azure cloud service](#).

## Prerequisites

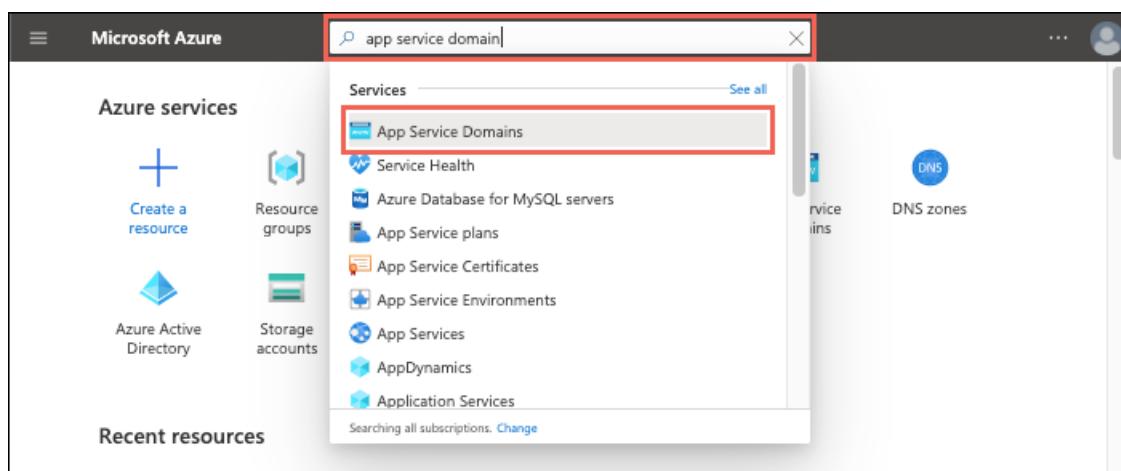
To complete this tutorial:

- [Create an App Service app](#), or use an app that you created for another tutorial. The app should be in an Azure Public region. At this time, Azure National Clouds are not supported.
- [Remove the spending limit on your subscription](#). You cannot buy App Service domains with free subscription credits.

## Buy an App Service domain

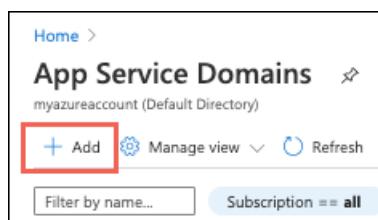
For pricing information on App Service domains, visit the [App Service Pricing page](#) and scroll down to App Service Domain.

1. Open the [Azure portal](#) and sign in with your Azure account.
2. In the search bar, search for and select **App Service Domains**.



The screenshot shows the Microsoft Azure portal interface. The search bar at the top has the text "app service domain". Below the search bar, the "Services" section is open, displaying a list of various Azure services. The "App Service Domains" option is highlighted with a red box. Other visible services include Service Health, Azure Database for MySQL servers, App Service plans, App Service Certificates, App Service Environments, App Services, AppDynamics, and Application Services. To the left, there's a sidebar with "Azure services" like Create a resource, Resource groups, and Storage accounts. At the bottom, there's a "Recent resources" section.

3. In the App Service Domains view, click Add.



The screenshot shows the "App Service Domains" view in the Azure portal. At the top, there's a breadcrumb navigation "Home > App Service Domains" and a context indicator "myazureaccount (Default Directory)". Below that is a toolbar with a "Add" button (highlighted with a red box), "Manage view", "Refresh", and search/filter options ("Filter by name..." and "Subscription == all").

4. Select [Click to try the newer version of the App Service Domains create experience](#).



Click to try the newer version of the App Service Domains create experience.



## Basics tab

1. In the **Basics** tab, configure the settings using the following table:

SETTING	DESCRIPTION
Subscription	The subscription to use to buy the domain.
Resource Group	The resource group to put the domain in. For example, the resource group your app is in.
Domain	Type the domain you want. For example, <code>contoso.com</code> . If the domain you want is not available, you can select from a list of suggestions of available domains, or try a different domain.

### NOTE

The following [top-level domains](#) are supported by App Service domains: `com`, `net`, `co.uk`, `org`, `nl`, `in`, `biz`, `org.uk`, and `co.in`.

2. When finished, click **Next: Contact information**.

## Contact information tab

1. Supply your information as required by [ICANN](#) for the domain registration.

It is important that you fill out all required fields with as much accuracy as possible. Incorrect data for contact information can result in failure to buy the domain.

2. When finished, click **Next: Advanced**.

## Advanced tab

1. In the **Advanced** tab, configure the optional settings:

SETTING	DESCRIPTION
Auto renewal	Enabled by default. Your App Service domain is registered to you at one-year increments. Auto renewal makes sure that your domain registration doesn't expire and that you retain ownership of the domain. Your Azure subscription is automatically charged the yearly domain registration fee at the time of renewal. To opt out, select <b>Disable</b> . If auto-renewal is disabled, you can <a href="#">renew it manually</a> .
Privacy protection	Enabled by default. Privacy protection hides your domain registration contact information from the WHOIS database. Privacy protection is already included in the yearly domain registration fee. To opt out, select <b>Disable</b> .

2. When finished, click **Next: Tags**.

## Finish

1. In the **Tags** tab, set the tags you want for your App Service domain. Tagging is not required for using App

Service domains, but is a [feature in Azure that helps you manage your resources](#).

2. Click **Next: Review + create**.
3. In the **Review + create** tab, review your domain order. When finished, click **Create**.

#### NOTE

App Service Domains use GoDaddy for domain registration and Azure DNS to host the domains. In addition to the yearly domain registration fee, usage charges for Azure DNS apply. For information, see [Azure DNS Pricing](#).

4. When the domain registration is complete, you see a **Go to resource** button. Select it to see its management page.

The screenshot shows the 'Overview' page for a deployment named 'Microsoft.Web-AppServiceDomains-Portal-b95943e9-bdea'. The main message is 'Your deployment is complete'. It provides deployment details: Deployment name: Microsoft.Web-AppServiceDomains-Portal-b95..., Start time: 11/27/2020, 4:14:40, Subscription: Visual Studio Ultimate with MSDN, Resource group: domain. There are sections for 'Deployment details' (with a download link) and 'Next steps'. A prominent red-bordered 'Go to resource' button is at the bottom.

You're now ready to assign an App Service app to this custom domain.

#### NOTE

Depending on the subscription type, a sufficient payment history may be required prior to creating an App Service Domain.

If you have made payments and are still running into this error, you can contact support and provide proof of payments.

## Prepare the app

To map a custom DNS name to a web app, the web app's [App Service plan](#) must be a paid tier (Shared, Basic, Standard, Premium, or Consumption for Azure Functions). In this step, you make sure that the App Service app is in the supported pricing tier.

#### 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.

## Navigate to the app in the Azure portal

1. From the top search bar, search for and select **App Services**.

The screenshot shows the Microsoft Azure portal's search interface. A search bar at the top contains the text "app services". Below the search bar, a list of services is displayed under the heading "Services". The "App Services" option is highlighted with a red box. Other listed services include Function App, Logic App (Preview), App Service Certificates, App Service Domains, App Service Environments, App Service plans, and Service Health.

2. Select the name of the app.

The screenshot shows the "App Services" management page. At the top, there are various navigation and filter options. Below this, a table displays one record: "my-demo-app". The table columns are Name, Status, Location, Pricing Tier, and App Service Plan. The "Name" column shows "my-demo-app" with a checkbox next to it, which is also highlighted with a red box. The status is "Running", location is "West Europe", pricing tier is "Free", and the app service plan is "myAppServicePlan".

You see the management page of the App Service app.

#### Check the pricing tier

1. In the left navigation of the app page, scroll to the **Settings** section and select **Scale up (App Service plan)**.

The screenshot shows the "cephalin320170403020701" App Service settings page. The left sidebar lists several settings sections: Deployment Center, Settings (Configuration, Container settings, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, TLS/SSL settings, Networking), and Scale up (App Service plan). The "Scale up (App Service plan)" section is highlighted with a red box.

2. The app's current tier is highlighted by a blue border. Check to make sure that the app is not in the F1 tier. Custom DNS is not supported in the F1 tier.



### Dev / Test

For less demanding workloads



### Production

For most production workloads



### Isolated

Advanced networking and scale

## Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

## Included hardware

Every instance of your App Service plan will include the following hardware configuration:

### Azure Compute Units (ACU)



Dedicated compute resources used to run applications deployed in the App...

### Memory



Memory available to run applications

3. If the App Service plan is not in the F1 tier, close the Scale up page and skip to [Buy the domain](#).

## Scale up the App Service plan

1. Select any of the non-free tiers (D1, B1, B2, B3, or any tier in the Production category). For additional options, click [See additional options](#).
2. Click **Apply**.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

### Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included features

Every app hosted on this App Service plan will have access to these features:

**Custom domains / SSL**

Configure and purchase custom domains with SNI SSL bindings

**Manual scale**

Up to 3 instances. Subject to availability.

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:

**Azure Compute Units (ACU)**

Dedicated compute resources used to run applications deployed in the App...

**Memory**

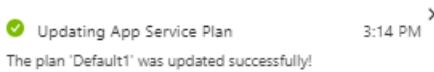
Memory per instance available to run applications deployed and running in...

**Storage**

10 GB disk storage shared by all apps deployed in the App Service plan.

**Apply**

When you see the following notification, the scale operation is complete.



### Map App Service domain to your app

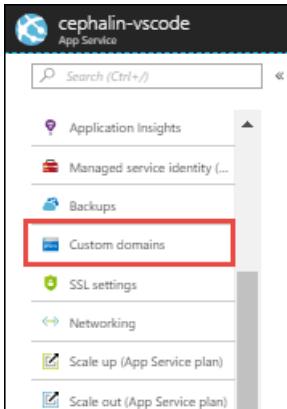
It's easy to map a hostname in your App Service domain to an App Service app, as long as it's in the same subscription. You map the App Service domain or any of its subdomain directly in your app, and Azure creates the necessary DNS records for you.

## NOTE

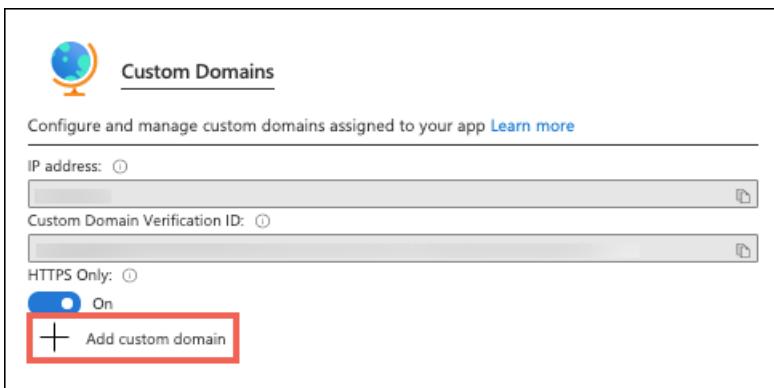
If the domain and the app are in different subscriptions, you map the App Service domain to the app just like [mapping an externally purchased domain](#). In this case, Azure DNS is the external domain provider, and you need to [add the required DNS records manually](#).

## Map the domain

1. In the left navigation of the app page, scroll to the **Settings** section and select **Custom domains**.



2. Select **Add custom domain**.



3. Type the App Service domain (such as [contoso.com](http://contoso.com)) or a subdomain (such as [www.contoso.com](http://www.contoso.com)) and click **Validate**.

## NOTE

If you made a typo in the App Service domain name, a verification error appears at the bottom of the page to tell you that you're missing some DNS records. You don't need to add these records manually for an App Service domain. Just make sure that you type the domain name correctly and click **Validate** again.

The screenshot shows a summary of DNS validation results:

- DNS propagation:** A note stating that depending on your DNS provider, it can take up to 48 hours for changes to propagate. It includes a link to [digwebinterface.com/](http://digwebinterface.com/).
- Hostname availability:** Indicated by a green checkmark.
- Domain ownership:** Indicated by a red exclamation mark. A note says to verify ownership by creating a CNAME or TXT record with specific values. Below is a table of record configurations:

Type	Host	Value
TXT	asuid.www or asuid.{subdomain}	0123456789ABCDEF0123456789ABCDEF01;{subdomain}
CNAME	www or {subdomain}	my-demo-app.azurewebsites.net

- Accept the Hostname record type and click **Add custom domain**.

The dialog is titled "Add custom domain" for "my-demo-app". It contains the following fields:

- Custom domain \***: www.contoso.com (highlighted with a red box)
- Validate** button (highlighted with a red box)
- Hostname record type**: CNAME (www.example.com or any subdomain) (highlighted with a red box)
- CNAME configuration** section:
  - A note about CNAME records being aliases for other domains.
  - Custom Domain Verification ID:** A redacted input field.
  - CNAME**: A redacted input field ending in .azurewebsites.net.
- Add custom domain** button (highlighted with a red box)

- It might take some time for the new custom domain to be reflected in the app's **Custom Domains** page. Refresh the browser to update the data.

Custom Domains

Configure and manage custom domains assigned to your app [Learn more](#)

IP address:

Custom Domain Verification ID:

HTTPS Only:  On

+ Add custom domain

Status Filter

All (2) Not Secure (1) Secure (1)

SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL Binding
!	www.contoso.com	Add binding
✓	my-demo-app.azurewebsites.net	

#### NOTE

A **Not Secure** label for your custom domain means that it's not yet bound to a TLS/SSL certificate. Any HTTPS request from a browser to your custom domain will receive an error or warning, depending on the browser. To add a TLS binding, see [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#).

## Test the custom domain

To test the custom domain, navigate to it in the browser.

## Renew the domain

The App Service domain you bought is valid for one year from the time of purchase. By default, the domain is configured to renew automatically by charging your payment method for the next year. You can manually renew your domain name.

If you want to turn off automatic renewal, or if you want to manually renew your domain, follow the steps here.

1. In the search bar, search for and select **App Service Domains**.

2. In the **App Service Domains** section, select the domain you want to configure.
3. From the left navigation of the domain, select **Domain renewal**. To stop renewing your domain automatically, select **Off**. The setting takes effect immediately.

contoso.com | Domain renewal

App Service Domain

Search (Cmd +/)

Overview

Access control (IAM)

Tags

Settings

Properties

Locks

Domain Management

Advanced Management portal

Hostname bindings

Domain renewal

Automation

Domain renewal

App Service domains can be set to auto renew to prevent expiration and unexpected domain ownership loss.

Auto renew domain: **Off**

⚠ Manual domain renewal can only be performed up to 90 days ahead of domain expiration and up to 18 days after domain expiration. You can enable domain auto-renewal policy to reduce the management overhead of this operations.

Expiration date: November 27, 2021, 10:15:02 AM GMT+1

Renew domain

#### NOTE

When navigating away from the page, disregard the "Your unsaved edits will be discarded" error by clicking **OK**.

To manually renew your domain, select **Renew domain**. However, this button is not active until [90 days before the domain's expiration](#).

If your domain renewal is successful, you receive an email notification within 24 hours.

## When domain expires

Azure deals with expiring or expired App Service domains as follows:

- If automatic renewal is disabled: 90 days before domain expiration, a renewal notification email is sent to you and the **Renew domain** button is activated in the portal.
- If automatic renewal is enabled: On the day after your domain expiration date, Azure attempts to bill you for the domain name renewal.
- If an error occurs during automatic renewal (for example, your card on file is expired), or if automatic renewal is disabled and you allow the domain to expire, Azure notifies you of the domain expiration and parks your domain name. You can [manually renew](#) your domain.
- On the 4th and 12th days day after expiration, Azure sends you additional notification emails. You can [manually renew](#) your domain. On the 5th day after expiration, DNS resolution stops for the expired domain.
- On the 19th day after expiration, your domain remains on hold but becomes subject to a redemption fee. You can call customer support to renew your domain name, subject to any applicable renewal and redemption fees.
- On the 25th day after expiration, Azure puts your domain up for auction with a domain name industry auction service. You can call customer support to renew your domain name, subject to any applicable renewal and redemption fees.
- On the 30th day after expiration, you're no longer able to redeem your domain.

## Manage custom DNS records

In Azure, DNS records for an App Service Domain are managed using [Azure DNS](#). You can add, remove, and update DNS records, just like for an externally purchased domain. To manage custom DNS records:

1. In the search bar, search for and select App Service Domains.

The screenshot shows the Microsoft Azure portal interface. In the top navigation bar, there is a search bar with the text "app service domain" entered. Below the search bar, under the heading "Services", the "App Service Domains" option is highlighted with a red box. To the left of the search bar, there is a sidebar titled "Azure services" with various icons and links like "Create a resource", "Resource groups", "Storage accounts", and "Recent resources". On the right side of the search results, there are two small cards: "Service Health" and "DNS zones".

2. In the App Service Domains section, select the domain you want to configure.

3. From the Overview page, select Manage DNS records.

The screenshot shows the Azure portal's "Overview" page for the "contoso.com" App Service Domain. The URL in the address bar is "contoso.com". At the top, there is a "Manage DNS records" button, which is highlighted with a red box. The page displays basic information such as the Resource group ("myResourceGroup"), Domain ("contoso.com"), Location ("Global"), Subscription ("Visual Studio Ultimate with MSDN"), and various status details like "Expiration date" (November 27, 2021) and "Auto renew" (Enabled). On the left, there is a navigation sidebar with links like "Overview", "Access control (IAM)", "Tags", "Settings", "Properties", and "Locks".

For information on how to edit DNS records, see [How to manage DNS Zones in the Azure portal](#).

## Cancel purchase (delete domain)

After you purchase the App Service Domain, you have five days to cancel your purchase for a full refund. After five days, you can delete the App Service Domain, but cannot receive a refund.

1. In the search bar, search for and select App Service Domains.

This screenshot is identical to the one above, showing the Microsoft Azure portal with the search bar containing "app service domain" and the "App Service Domains" item selected in the search results. The layout and sidebar elements are the same.

2. In the App Service Domains section, select the domain you want to configure.

3. In the domain's left navigation, select Locks.

A delete lock has been created for your domain. As long as a delete lock exists, you can't delete the App

Service domain.

4. Click **Delete** to remove the lock.
5. In the domain's left navigation, select **Overview**.
6. If the cancellation period on the purchased domain has not elapsed, select **Cancel purchase**. Otherwise, you see a **Delete** button instead. To delete the domain without a refund, select **Delete**.

The screenshot shows the Azure portal interface for managing an App Service Domain. The top navigation bar includes a search bar, 'Manage DNS records', 'FAQs', and a 'Cancel purchase' button, which is highlighted with a red box. The main content area is titled 'Overview' (also highlighted with a red box) and displays domain details:

Essentials	
Resource group ( <a href="#">change</a> ) myResourceGroup	Domain contoso.com
Location Global	Expiration date November 27, 2021, 10:15:02 AM GMT+1
Subscription ( <a href="#">change</a> ) Visual Studio Ultimate with MSDN	Auto renew Enabled
Subscription ID 00000000-0000-0000-000000000000	Privacy protection Enabled

7. Confirm the operation by selecting **Yes**.

After the operation is complete, the domain is released from your subscription and available for anyone to purchase again.

## Direct default URL to a custom directory

By default, App Service directs web requests to the root directory of your app code. To direct them to a subdirectory, such as `public`, see [Redirect to a custom directory](#).

## Next steps

Learn how to bind a custom TLS/SSL certificate to App Service.

[Secure a custom DNS name with a TLS binding in Azure App Service](#)

# Configure a custom domain name in Azure App Service with Traffic Manager integration

11/2/2021 • 5 minutes to read • [Edit Online](#)

## NOTE

For Cloud Services, see [Configuring a custom domain name for an Azure cloud service](#).

When you use [Azure Traffic Manager](#) to load balance traffic to [Azure App Service](#), the App Service app can be accessed using <traffic-manager-endpoint>.trafficmanager.net. You can assign a custom domain name, such as www.contoso.com, with your App Service app in order to provide a more recognizable domain name for your users.

This article shows you how to configure a custom domain name with an App Service app that's integrated with [Traffic Manager](#).

## NOTE

Only [CNAME](#) records are supported when you configure a domain name using the Traffic Manager endpoint. Because A records are not supported, a root domain mapping, such as contoso.com is also not supported.

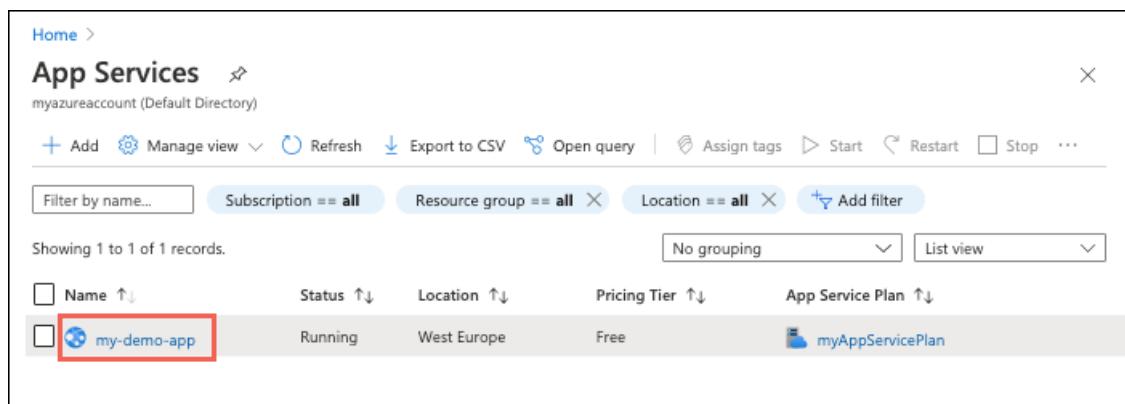
## Prepare the app

To map a custom DNS name to an app that's integrated with Azure Traffic Manager, the web app's [App Service plan](#) must be in **Standard** tier or higher. In this step, you make sure that the App Service app is in the supported pricing tier.

### Check the pricing tier

In the [Azure portal](#), search for and select [App Services](#).

On the [App Services](#) page, select the name of your Azure app.



The screenshot shows the Azure App Services blade. At the top, there are navigation links for Home, App Services, and a search bar. Below the search bar are filter options: Filter by name..., Subscription == all, Resource group == all, Location == all, and Add filter. The main table displays one record: my-demo-app. The columns are Name, Status, Location, Pricing Tier, and App Service Plan. The 'Name' column contains a link to 'my-demo-app' which is highlighted with a red box. The status is 'Running', location is 'West Europe', pricing tier is 'Free', and the app service plan is 'myAppServicePlan'.

Name	Status	Location	Pricing Tier	App Service Plan
my-demo-app	Running	West Europe	Free	myAppServicePlan

In the left navigation of the app page, select **Scale up (App Service plan)**.

 cephalin320170403020701  
App Service

---

Search (Ctrl+ /)

 Deployment Center

**Settings**

---

 Configuration

 Container settings

 Authentication / Authorization

 Application Insights

 Identity

 Backups

 Custom domains

 TLS/SSL settings

 Networking

 Scale up (App Service plan)

The app's current tier is highlighted by a blue border. Check to make sure that the app is in **Standard** tier or above (any tier in the **Production** or **Isolated** category). If yes, close the **Scale up** page and skip to [Create the CNAME mapping](#).



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

## Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

## Included hardware

Every instance of your App Service plan will include the following hardware configuration:

### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...

### Memory

Memory available to run applications

## Scale up the App Service plan

If you need to scale up your app, select any of the pricing tiers in the **Production** category. For additional options, click [See additional options](#).

Click **Apply**.

## Create Traffic Manager endpoint

Following the steps at [Add or Delete Endpoints](#), add your App Service app as an endpoint in your Traffic Manager profile.

Once your App Service app is in a supported pricing tier, it shows up in the list of available App Service targets when you add the endpoint. If your app isn't listed, [verify the pricing tier of your app](#).

## Create the CNAME mapping

### NOTE

To configure an [App Service domain that you purchased](#), skip this section and go to [Enable custom domain](#).

#### NOTE

You can use Azure DNS to configure a custom DNS name for Azure App Service. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

1. Sign in to the website of your domain provider.
2. Find the page for managing DNS records. Every domain provider has its own DNS records interface, so consult the provider's documentation. Look for areas of the site labeled **Domain Name, DNS, or Name Server Management**.

Often, you can find the DNS records page by viewing your account information and then looking for a link such as **My domains**. Go to that page, and then look for a link that's named something like **Zone file, DNS Records, or Advanced configuration**.

The following screenshot is an example of a DNS records page:

The screenshot shows a table titled "Records" with the following data:

Type	Name	Value	TTL
NS	@	ns31.domaincontrol.com	1 Hour
NS	@	ns32.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns31.domaincontrol.com.	600 seconds

An "ADD" button is located at the bottom right of the table.

3. In the example screenshot, select **Add** to create a record. Some providers have different links to add different record types. Again, consult the provider's documentation.

#### NOTE

For certain providers, such as GoDaddy, changes to DNS records don't become effective until you select a separate **Save Changes** link.

While the specifics of each domain provider vary, you map *from* a [non-root custom domain name](#) (such as [www.contoso.com](#)) *to* the Traffic Manager domain name ([contoso.trafficmanager.net](#)) that's integrated with your app.

#### NOTE

If a record is already in use and you need to preemptively bind your apps to it, you can create an additional CNAME record. For example, to preemptively bind [www.contoso.com](#) to your app, create a CNAME record from [awverify.www](#) to [contoso.trafficmanager.net](#). You can then add "www.contoso.com" to your app without the need to change the "www" CNAME record. For more information, see [Migrate an active DNS name to Azure App Service](#).

Once you have finished adding or modifying DNS records at your domain provider, save the changes.

#### What about root domains?

Since Traffic Manager only supports custom domain mapping with CNAME records, and because DNS standards don't support CNAME records for mapping root domains (for example, [contoso.com](#)), Traffic Manager doesn't support mapping to root domains. To work around this issue, use a URL redirect from at the app level. In ASP.NET Core, for example, you can use [URL Rewriting](#). Then, use Traffic Manager to load balance the subdomain

([www.contoso.com](http://www.contoso.com)). Another approach is you can [create an alias record for your domain name apex to reference an Azure Traffic Manager profile](#). An example is contoso.com. Instead of using a redirecting service, you can configure Azure DNS to reference a Traffic Manager profile directly from your zone.

For high availability scenarios, you can implement a load-balancing DNS setup without Traffic Manager by creating multiple *A records* that point from the root domain to each app copy's IP address. Then, [map the same root domain to all the app copies](#). Since the same domain name cannot be mapped to two different apps in the same region, this setup only works when your app copies are in different regions.

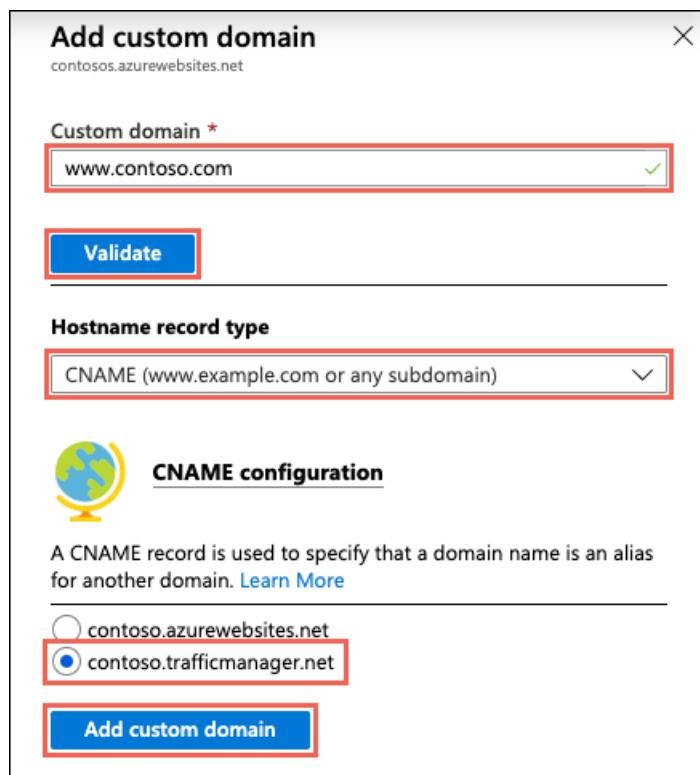
## Enable custom domain

After the records for your domain name have propagated, use the browser to verify that your custom domain name resolves to your App Service app.

### NOTE

It can take some time for your CNAME to propagate through the DNS system. You can use a service such as <https://www.digwebinterface.com/> to verify that the CNAME is available.

1. Once domain resolution succeeds, go back to your app page in the [Azure portal](#)
2. From the left navigation, select **Custom domains > Add hostname**.
3. Type the custom domain name that you mapped earlier and select **Validate**.
4. Make sure that **Hostname record type** is set to **CNAME (www.example.com or any subdomain)**.
5. Since the App Service app is now integrated with a Traffic Manager endpoint, you should see the Traffic Manager domain name under **CNAME configuration**. Select it and click **Add custom domain**.



## Next steps

[Secure a custom DNS name with an TLS/SSL binding in Azure App Service](#)

# Migrate an active DNS name to Azure App Service

11/2/2021 • 5 minutes to read • [Edit Online](#)

This article shows you how to migrate an active DNS name to [Azure App Service](#) without any downtime.

When you migrate a live site and its DNS domain name to App Service, that DNS name is already serving live traffic. You can avoid downtime in DNS resolution during the migration by binding the active DNS name to your App Service app preemptively.

If you're not worried about downtime in DNS resolution, see [Map an existing custom DNS name to Azure App Service](#).

## Prerequisites

To complete this how-to:

- [Make sure that your App Service app is not in FREE tier.](#)

## Bind the domain name preemptively

When you bind a custom domain preemptively, you accomplish both of the following before making any changes to your existing DNS records:

- Verify domain ownership
- Enable the domain name for your app

When you finally migrate your custom DNS name from the old site to the App Service app, there will be no downtime in DNS resolution.

### Access DNS records with domain provider

#### NOTE

You can use Azure DNS to configure a custom DNS name for Azure App Service. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

1. Sign in to the website of your domain provider.
2. Find the page for managing DNS records. Every domain provider has its own DNS records interface, so consult the provider's documentation. Look for areas of the site labeled **Domain Name**, **DNS**, or **Name Server Management**.

Often, you can find the DNS records page by viewing your account information and then looking for a link such as **My domains**. Go to that page, and then look for a link that's named something like **Zone file**, **DNS Records**, or **Advanced configuration**.

The following screenshot is an example of a DNS records page:

## Records

Last updated 6/18/2018 3:40 PM

Type	Name	Value	TTL
NS	@	ns31.domaincontrol.com	1 Hour
NS	@	ns32.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns31.domaincontrol.com.	600 seconds

[ADD](#)

- In the example screenshot, select **Add** to create a record. Some providers have different links to add different record types. Again, consult the provider's documentation.

### NOTE

For certain providers, such as GoDaddy, changes to DNS records don't become effective until you select a separate **Save Changes** link.

## Get domain verification ID

Get the domain verification ID for your app by following the steps at [Get domain verification ID](#).

### Create domain verification record

To verify domain ownership, add a TXT record for domain verification. The hostname for the TXT record depends on the type of DNS record type you want to map. See the following table (@ typically represents the root domain):

DNS RECORD EXAMPLE	TXT HOST	TXT VALUE
@ (root)	<i>asuid</i>	Domain verification ID for your app
www (sub)	<i>asuid.www</i>	Domain verification ID for your app
* (wildcard)	<i>asuid</i>	Domain verification ID for your app

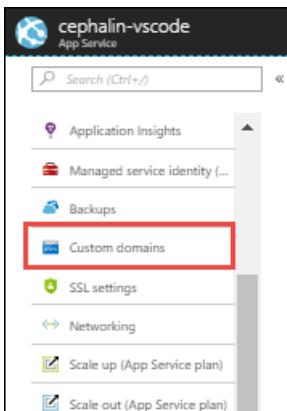
In your DNS records page, note the record type of the DNS name you want to migrate. App Service supports mappings from CNAME and A records.

### NOTE

Wildcard (\*) records won't validate subdomains with an existing CNAME's record. You may need to explicitly create a TXT record for each subdomain.

## Enable the domain for your app

- In the [Azure portal](#), in the left navigation of the app page, select **Custom domains**.



2. In the **Custom domains** page, select **Add custom domain**.

A screenshot of the 'Custom Domains' configuration page. It features a globe icon and the title 'Custom Domains'. Below the title, there's a sub-header 'Configure and manage custom domains assigned to your app' with a 'Learn more' link. There are three input fields: 'IP address' (with a placeholder '123.45.67.89'), 'Custom Domain Verification ID' (with a placeholder '00000000-0000-0000-0000-000000000000'), and 'HTTPS Only' (set to 'On'). At the bottom left is a red-bordered 'Add custom domain' button.

3. Type the fully qualified domain name you want to migrate, that corresponds to the TXT record you create, such as `contoso.com`, `www.contoso.com`, or `*.contoso.com`. Select **Validate**.

The **Add custom domain** button is activated.

4. Make sure that **Hostname record type** is set to the DNS record type you want to migrate. Select **Add hostname**.

A screenshot of the 'Add custom domain' dialog for the 'my-demo-app'. It shows a 'Custom domain \*' field containing 'www.contoso.com' (highlighted with a red box). Below it is a 'Validate' button in a blue box. Under 'Hostname record type', there's a dropdown menu set to 'CNAME (www.example.com or any subdomain)' (highlighted with a red box). At the bottom, there's a 'CNAME configuration' section with a globe icon, a note about CNAME records, and a 'Custom Domain Verification ID' field with a placeholder value (highlighted with a red box). Finally, there's an 'Add custom domain' button at the bottom.

It might take some time for the new hostname to be reflected in the app's **Custom domains** page. Try refreshing the browser to update the data.

The screenshot shows the 'Custom Domains' section of the Azure portal. It includes fields for 'IP address' and 'Custom Domain Verification ID'. A toggle switch for 'HTTPS Only' is set to 'On'. A status filter bar at the bottom shows 'All (2)', 'Not Secure (1)', and 'Secure (1)'. Below this, a table lists 'ASSIGNED CUSTOM DOMAINS' with two rows: one for 'Not Secure' (www.contoso.com) and one for 'Secure' (my-demo-app.azurewebsites.net). The 'Secure' row has an 'Add binding' link.

Your custom DNS name is now enabled in your Azure app.

## Remap the active DNS name

The only thing left to do is remapping your active DNS record to point to App Service. Right now, it still points to your old site.

### Copy the app's IP address (A record only)

If you are remapping a CNAME record, skip this section.

To remap an A record, you need the App Service app's external IP address, which is shown in the **Custom domains** page.

In the **Custom domains** page, copy the app's IP address.

This screenshot is identical to the one above, but the 'IP address' input field is highlighted with a red rectangle to draw attention to it.

## Update the DNS record

Back in the DNS records page of your domain provider, select the DNS record to remap.

For the `contoso.com` root domain example, remap the A or CNAME record like the examples in the following table:

FQDN EXAMPLE	RECORD TYPE	HOST	VALUE
contoso.com (root)	A	@	IP address from <a href="#">Copy the app's IP address</a>
www.contoso.com (sub)	CNAME	www	<appname>.azurewebsites.net
*.contoso.com (wildcard)	CNAME	*	<appname>.azurewebsites.net

Save your settings.

DNS queries should start resolving to your App Service app immediately after DNS propagation happens.

## Migrate domain from another app

You can migrate an active custom domain in Azure, between subscriptions or within the same subscription. However, such a migration without downtime requires the source app and the target app are assigned the same custom domain at a certain time. Therefore, you need to make sure that the two apps are not deployed to the same deployment unit (internally known as a webspace). A domain name can be assigned to only one app in each deployment unit.

You can find the deployment unit for your app by looking at the domain name of the FTP/S URL

<deployment-unit>.ftp.azurewebsites.windows.net. Check and make sure the deployment unit is different between the source app and the target app. The deployment unit of an app is determined by the [App Service plan](#) it's in. It's selected randomly by Azure when you create the plan and can't be changed. Azure only makes sure two plans are in the same deployment unit when you [create them in the same resource group and the same region](#), but it doesn't have any logic to make sure plans are in different deployment units. The only way for you to create a plan in a different deployment unit is to keep creating a plan in a new resource group or region until you get a different deployment unit.

## Next steps

Learn how to bind a custom TLS/SSL certificate to App Service.

[Secure a custom DNS name with a TLS binding in Azure App Service](#)

# Configure your App Service or Azure Functions app to use Azure AD login

11/2/2021 • 10 minutes to read • [Edit Online](#)

This article shows you how to configure authentication for Azure App Service or Azure Functions so that your app signs in users with the [Microsoft identity platform](#) (Azure AD) as the authentication provider.

The App Service Authentication feature can automatically create an app registration with the Microsoft identity platform. You can also use a registration that you or a directory admin creates separately.

- [Create a new app registration automatically](#)
- [Use an existing registration created separately](#)

## NOTE

The option to create a new registration is not available for government clouds. Instead, [define a registration separately](#).

## Option 1: Create a new app registration automatically

This option is designed to make enabling authentication simple and requires just a few clicks.

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Microsoft** in the identity provider dropdown. The option to create a new registration is selected by default. You can change the name of the registration or the supported account types.

A client secret will be created and stored as a slot-sticky [application setting](#) named `MICROSOFT_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. (Optional) Click **Next: Permissions** and add any scopes needed by the application. These will be added to the app registration, but you can also change them later.
6. Click **Add**.

You're now ready to use the Microsoft identity platform for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

For an example of configuring Azure AD login for a web app that accesses Azure Storage and Microsoft Graph, see [this tutorial](#).

## Option 2: Use an existing registration created separately

You can also manually register your application for the Microsoft identity platform, customizing the registration and configuring App Service Authentication with the registration details. This is useful, for example, if you want to use an app registration from a different Azure AD tenant than the one your application is in.

## Create an app registration in Azure AD for your App Service app

First, you will create your app registration. As you do so, collect the following information which you will need later when you configure the authentication in the App Service app:

- Client ID
- Tenant ID
- Client secret (optional)
- Application ID URI

To register the app, perform the following steps:

1. Sign in to the [Azure portal](#), search for and select **App Services**, and then select your app. Note your app's URL. You'll use it to configure your Azure Active Directory app registration.
2. From the portal menu, select **Azure Active Directory**, then go to the **App registrations** tab and select **New registration**.
3. In the **Register an application** page, enter a **Name** for your app registration.
4. In **Redirect URI**, select **Web** and type `<app-url>/.auth/login/aad/callback`. For example, `https://contoso.azurewebsites.net/.auth/login/aad/callback`.
5. Select **Register**.
6. After the app registration is created, copy the **Application (client) ID** and the **Directory (tenant) ID** for later.
7. Select **Authentication**. Under **Implicit grant and hybrid flows**, enable **ID tokens** to allow OpenID Connect user sign-ins from App Service. Select **Save**.
8. (Optional) Select **Branding**. In **Home page URL**, enter the URL of your App Service app and select **Save**.
9. Select **Expose an API**, and click **Set** next to "Application ID URI". This value uniquely identifies the application when it is used as a resource, allowing tokens to be requested that grant access. It is used as a prefix for scopes you create.

For a single-tenant app, you can use the default value, which is in the form the form `api://<application-client-id>`. You can also specify a more readable URI like `https://contoso.com/api` based on one of the verified domains for your tenant. For a multi-tenant app, you must provide a custom URI. To learn more about accepted formats for App ID URIs, see the [app registrations best practices reference](#).

Once you have entered the value, click **Save**.

10. Select **Add a scope**.
  - a. In **Add a scope**, the **Application ID URI** is the value you set in a previous step. Select **Save and continue**.
  - b. In **Scope name**, enter `user_impersonation`.
  - c. In the text boxes, enter the consent scope name and description you want users to see on the consent page. For example, enter `Access <application-name>`.
  - d. Select **Add scope**.
11. (Optional) To create a client secret, select **Certificates & secrets > Client secrets > New client**

**secret.** Enter a description and expiration and select **Add**. Copy the client secret value shown in the page. It won't be shown again.

12. (Optional) To add multiple **Reply URLs**, select **Authentication**.

#### Enable Azure Active Directory in your App Service app

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Microsoft** in the identity provider dropdown.
4. For **App registration type**, you can choose to **Pick an existing app registration in this directory** which will automatically gather the necessary app information. If your registration is from another tenant or you do not have permission to view the registration object, choose **Provide the details of an existing app registration**. For this option, you will need to fill in the following configuration details:

FIELD	DESCRIPTION
Application (client) ID	Use the <b>Application (client) ID</b> of the app registration.
Client Secret	Use the client secret you generated in the app registration. With a client secret, hybrid flow is used and the App Service will return access and refresh tokens. When the client secret is not set, implicit flow is used and only an ID token is returned. These tokens are sent by the provider and stored in the EasyAuth token store.
Issuer Url	Use <code>&lt;authentication-endpoint&gt;/&lt;tenant-id&gt;/v2.0</code> , and replace <code>&lt;authentication-endpoint&gt;</code> with the <a href="#">authentication endpoint for your cloud environment</a> (e.g., "https://login.microsoftonline.com" for global Azure), also replacing <code>&lt;tenant-id&gt;</code> with the <b>Directory (tenant) ID</b> in which the app registration was created. This value is used to redirect users to the correct Azure AD tenant, as well as to download the appropriate metadata to determine the appropriate token signing keys and token issuer claim value for example. For applications that use Azure AD v1 and for Azure Functions apps, omit <code>/v2.0</code> in the URL.
Allowed Token Audiences	If this is a cloud or server app and you want to allow authentication tokens from a web app, add the <b>Application ID URI</b> of the web app here. The configured <b>Client ID</b> is <i>always</i> implicitly considered to be an allowed audience.

The client secret will be stored as a slot-sticky [application setting](#) named `MICROSOFT_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

5. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

## 6. Click Add.

You're now ready to use the Microsoft identity platform for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

# Configure client apps to access your App Service

In the prior section, you registered your App Service or Azure Function to authenticate users. This section explains how to register native client or daemon apps so that they can request access to APIs exposed by your App Service on behalf of users or themselves. Completing the steps in this section is not required if you only wish to authenticate users.

## Native client application

You can register native clients to request access to your App Service app's APIs on behalf of a signed in user.

1. In the [Azure portal](#), select Active Directory > App registrations > New registration.
2. In the Register an application page, enter a Name for your app registration.
3. In Redirect URI, select Public client (mobile & desktop) and type the URL  
`<app-url>/auth/login/aad/callback`. For example,  
`https://contoso.azurewebsites.net/.auth/login/aad/callback`.

### NOTE

For a Microsoft Store application, use the [package SID](#) as the URI instead.

4. Select Create.
5. After the app registration is created, copy the value of Application (client) ID.
6. Select API permissions > Add a permission > My APIs.
7. Select the app registration you created earlier for your App Service app. If you don't see the app registration, make sure that you've added the `user_impersonation` scope in [Create an app registration in Azure AD for your App Service app](#).
8. Under Delegated permissions, select `user_impersonation`, and then select Add permissions.

You have now configured a native client application that can request access to your App Service app on behalf of a user.

## Daemon client application (service-to-service calls)

Your application can acquire a token to call a Web API hosted in your App Service or Function app on behalf of itself (not on behalf of a user). This scenario is useful for non-interactive daemon applications that perform tasks without a logged in user. It uses the standard OAuth 2.0 [client credentials](#) grant.

1. In the [Azure portal](#), select Active Directory > App registrations > New registration.
2. In the Register an application page, enter a Name for your daemon app registration.
3. For a daemon application, you don't need a Redirect URI so you can keep that empty.
4. Select Create.
5. After the app registration is created, copy the value of Application (client) ID.
6. Select Certificates & secrets > New client secret > Add. Copy the client secret value shown in the page. It won't be shown again.

You can now [request an access token using the client ID and client secret](#) by setting the `resource` parameter to

the **Application ID URI** of the target app. The resulting access token can then be presented to the target app using the standard [OAuth 2.0 Authorization header](#), and App Service Authentication / Authorization will validate and use the token as usual to now indicate that the caller (an application in this case, not a user) is authenticated.

At present, this allows *any* client application in your Azure AD tenant to request an access token and authenticate to the target app. If you also want to enforce *authorization* to allow only certain client applications, you must perform some additional configuration.

1. [Define an App Role](#) in the manifest of the app registration representing the App Service or Function app you want to protect.
2. On the app registration representing the client that needs to be authorized, select **API permissions > Add a permission > My APIs**.
3. Select the app registration you created earlier. If you don't see the app registration, make sure that you've [added an App Role](#).
4. Under **Application permissions**, select the App Role you created earlier, and then select **Add permissions**.
5. Make sure to click **Grant admin consent** to authorize the client application to request the permission.
6. Similar to the previous scenario (before any roles were added), you can now [request an access token](#) for the same target `resource`, and the access token will include a `roles` claim containing the App Roles that were authorized for the client application.
7. Within the target App Service or Function app code, you can now validate that the expected roles are present in the token (this is not performed by App Service Authentication / Authorization). For more information, see [Access user claims](#).

You have now configured a daemon client application that can access your App Service app using its own identity.

## Best practices

Regardless of the configuration you use to set up authentication, the following best practices will keep your tenant and applications more secure:

- Give each App Service app its own permissions and consent.
- Configure each App Service app with its own registration.
- Avoid permission sharing between environments by using separate app registrations for separate deployment slots. When testing new code, this practice can help prevent issues from affecting the production app.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)
- [Tutorial: Authenticate and authorize users in a web app that accesses Azure Storage and Microsoft Graph](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

# Configure your App Service or Azure Functions app to use Facebook login

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows how to configure Azure App Service or Azure Functions to use Facebook as an authentication provider.

To complete the procedure in this article, you need a Facebook account that has a verified email address and a mobile phone number. To create a new Facebook account, go to [facebook.com](https://facebook.com).

## Register your application with Facebook

1. Go to the [Facebook Developers](#) website and sign in with your Facebook account credentials.

If you don't have a Facebook for Developers account, select **Get Started** and follow the registration steps.

2. Select **My Apps > Add New App**.

3. In **Display Name** field:

- a. Type a unique name for your app.
- b. Provide your **Contact Email**.
- c. Select **Create App ID**.
- d. Complete the security check.

The developer dashboard for your new Facebook app opens.

4. Select **Dashboard > Facebook Login > Set up > Web**.

5. In the left navigation under **Facebook Login**, select **Settings**.

6. In the **Valid OAuth redirect URIs** field, enter

`https://<app-name>.azurewebsites.net/.auth/login/facebook/callback`. Remember to replace `<app-name>` with the name of your Azure App Service app.

7. Select **Save Changes**.

8. In the left pane, select **Settings > Basic**.

9. In the **App Secret** field, select **Show**. Copy the values of **App ID** and **App Secret**. You use them later to configure your App Service app in Azure.

### IMPORTANT

The app secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

10. The Facebook account that you used to register the application is an administrator of the app. At this point, only administrators can sign in to this application.

To authenticate other Facebook accounts, select **App Review** and enable **Make <your-app-name> public** to enable the general public to access the app by using Facebook authentication.

## Add Facebook information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Facebook** in the identity provider dropdown. Paste in the App ID and App Secret values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named

`FACEBOOK_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. (Optional) Click **Next: Scopes** and add any scopes needed by the application. These will be requested at login time for browser-based flows.
6. Click **Add**.

You're now ready to use Facebook for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

# Configure your App Service or Azure Functions app to use Google login

11/2/2021 • 2 minutes to read • [Edit Online](#)

This topic shows you how to configure Azure App Service or Azure Functions to use Google as an authentication provider.

To complete the procedure in this topic, you must have a Google account that has a verified email address. To create a new Google account, go to [accounts.google.com](#).

## Register your application with Google

1. Follow the Google documentation at [Google Sign-In for server-side apps](#) to create a client ID and client secret. There's no need to make any code changes. Just use the following information:
  - For **Authorized JavaScript Origins**, use `https://<app-name>.azurewebsites.net` with the name of your app in `<app-name>`.
  - For **Authorized Redirect URI**, use `https://<app-name>.azurewebsites.net/.auth/login/google/callback`.
2. Copy the App ID and the App secret values.

### IMPORTANT

The App secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

## Add Google information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Google** in the identity provider dropdown. Paste in the App ID and App Secret values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named `GOOGLE_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Click **Add**.

[Note] > For adding scope: You can define what permissions your application has in the provider's registration portal. The app can request scopes at login time which leverage these permissions.

You are now ready to use Google for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

# Configure your App Service or Azure Functions app to use Twitter login

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows how to configure Azure App Service or Azure Functions to use Twitter as an authentication provider.

To complete the procedure in this article, you need a Twitter account that has a verified email address and phone number. To create a new Twitter account, go to [twitter.com](#).

## Register your application with Twitter

1. Sign in to the [Azure portal](#) and go to your application. Copy your **URL**. You'll use it to configure your Twitter app.
2. Go to the [Twitter Developers](#) website, sign in with your Twitter account credentials, and select **Create an app**.
3. Enter the **App name** and the **Application description** for your new app. Paste your application's **URL** into the **Website URL** field. In the **Callback URLs** section, enter the HTTPS URL of your App Service app and append the path `/auth/login/twitter/callback`. For example,  
`https://contoso.azurewebsites.net/.auth/login/twitter/callback`.
4. At the bottom of the page, type at least 100 characters in **Tell us how this app will be used**, then select **Create**. Click **Create** again in the pop-up. The application details are displayed.
5. Select the **Keys and Access Tokens** tab.

Make a note of these values:

- API key
- API secret key

### IMPORTANT

The API secret key is an important security credential. Do not share this secret with anyone or distribute it with your app.

## Add Twitter information to your application

1. Sign in to the [Azure portal](#) and navigate to your app.
2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **Twitter** in the identity provider dropdown. Paste in the `API key` and `API secret key` values that you obtained previously.

The secret will be stored as a slot-sticky [application setting](#) named

`TWITTER_PROVIDER_AUTHENTICATION_SECRET`. You can update that setting later to use [Key Vault references](#) if you wish to manage the secret in Azure Key Vault.

4. If this is the first identity provider configured for the application, you will also be prompted with an **App**

**Service authentication settings** section. Otherwise, you may move on to the next step.

These options determine how your application responds to unauthenticated requests, and the default selections will redirect all requests to log in with this new provider. You can change customize this behavior now or adjust these settings later from the main **Authentication** screen by choosing **Edit** next to **Authentication settings**. To learn more about these options, see [Authentication flow](#).

5. Click **Add**.

You're now ready to use Twitter for authentication in your app. The provider will be listed on the **Authentication** screen. From there, you can edit or delete this provider configuration.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

# Configure your App Service or Azure Functions app to login using an OpenID Connect provider (Preview)

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to configure Azure App Service or Azure Functions to use a custom authentication provider that adheres to the [OpenID Connect specification](#). OpenID Connect (OIDC) is an industry standard used by many identity providers (IDPs). You do not need to understand the details of the specification in order to configure your app to use an adherent IDP.

You can configure your app to use one or more OIDC providers. Each must be given a unique alphanumeric name in the configuration, and only one can serve as the default redirect target.

## Register your application with the identity provider

Your provider will require you to register the details of your application with it. One of these steps involves specifying a redirect URI. This redirect URI will be of the form `<app-url>/auth/login/<provider-name>/callback`. Each identity provider should provide more instructions on how to complete these steps.

### NOTE

Some providers may require additional steps for their configuration and how to use the values they provide. For example, Apple provides a private key which is not itself used as the OIDC client secret, and you instead must use it craft a JWT which is treated as the secret you provide in your app config (see the "Creating the Client Secret" section of the [Sign in with Apple documentation](#))

You will need to collect a **client ID** and **client secret** for your application.

### IMPORTANT

The client secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

Additionally, you will need the OpenID Connect metadata for the provider. This is often exposed via a [configuration metadata document](#), which is the provider's Issuer URL suffixed with `/.well-known/openid-configuration`. Gather this configuration URL.

If you are unable to use a configuration metadata document, you will need to gather the following values separately:

- The issuer URL (sometimes shown as `issuer`)
- The [OAuth 2.0 Authorization endpoint](#) (sometimes shown as `authorization_endpoint`)
- The [OAuth 2.0 Token endpoint](#) (sometimes shown as `token_endpoint`)
- The URL of the [OAuth 2.0 JSON Web Key Set](#) document (sometimes shown as `jwks_uri`)

## Add provider information to your application

1. Sign in to the [Azure portal] and navigate to your app.

2. Select **Authentication** in the menu on the left. Click **Add identity provider**.
3. Select **OpenID Connect** in the identity provider dropdown.
4. Provide the unique alphanumeric name selected earlier for **OpenID provider name**.
5. If you have the URL for the **metadata document** from the identity provider, provide that value for **Metadata URL**. Otherwise, select the **Provide endpoints separately** option and put each URL gathered from the identity provider in the appropriate field.
6. Provide the earlier collected **Client ID** and **Client Secret** in the appropriate fields.
7. Specify an application setting name for your client secret. Your client secret will be stored as an app setting to ensure secrets are stored in a secure fashion. You can update that setting later to use **Key Vault references** if you wish to manage the secret in Azure Key Vault.
8. Press the **Add** button to finish setting up the identity provider.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

# Configure your App Service or Azure Functions app to sign in using a Sign in with Apple provider (Preview)

11/2/2021 • 6 minutes to read • [Edit Online](#)

This article shows you how to configure Azure App Service or Azure Functions to use Sign in with Apple as an authentication provider.

To complete the procedure in this article, you must have enrolled in the Apple developer program. To enroll in the Apple developer program, go to [developer.apple.com/programs/enroll](https://developer.apple.com/programs/enroll).

## Caution

Enabling Sign in with Apple will disable management of the App Service Authentication / Authorization feature for your application through some clients, such as the Azure portal, Azure CLI, and Azure PowerShell. The feature relies on a new API surface which, during preview, is not yet accounted for in all management experiences.

## Create an application in the Apple Developer portal

You'll need to create an App ID and a service ID in the Apple Developer portal.

1. On the Apple Developer portal, go to **Certificates, Identifiers, & Profiles**.
2. On the **Identifiers** tab, select the (+) button.
3. On the **Register a New Identifier** page, choose **App IDs** and select **Continue**. (App IDs include one or more Service IDs.)

### Register a New Identifier

[Continue](#)

**App IDs**  
Register an App ID to enable your app to access available services and identify your app in a provisioning profile. You can enable app services when you create an App ID or modify these settings later.

**Services IDs**  
For each website that uses Sign in with Apple, register a services identifier (Services ID), configure your domain and return URL, and create an associated private key.

4. On the **Register an App ID** page, provide a description and a bundle ID, and select **Sign in with Apple** from the capabilities list. Then select **Continue**. Take note of your **App ID Prefix (Team ID)** from this step, you'll need it later.

### Register an App ID

[Back](#) [Continue](#)

Platform  iOS, tvOS, watchOS  macOS

App ID Prefix E88K2FF6LU (Team ID)

Description easy auth test sign in with apple

Bundle ID  Explicit  Wildcard com.microsoft.easyauthtest.client

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (\*).

### Capabilities

ENABLED NAME  
 Access WiFi Information

5. Review the app registration information and select **Register**.

6. Again, on the **Identifiers** tab, select the (+) button.

## Certificates, Identifiers & Profiles

Certificates	Identifiers <a href="#">+</a>
Identifiers	
Devices	
Profiles	
Keys	
More	

7. On the **Register a New Identifier** page, choose **Services IDs** and select **Continue**.

### Register a New Identifier

[Continue](#)

**App IDs**  
Register an App ID to enable your app to access available services and identify your app in a provisioning profile. You can enable app services when you create an App ID or modify these settings later.

**Services IDs**  
For each website that uses Sign in with Apple, register a services identifier (Services ID), configure your domain and return URL, and create an associated private key.

8. On the **Register a Services ID** page, provide a description and an identifier. The description is what will be shown to the user on the consent screen. The identifier will be your client ID used in configuring the Apple provider with your app service. Then select **Configure**.

### Register a Services ID

[Back](#) [Configure](#)

Description easy auth test sign in with apple

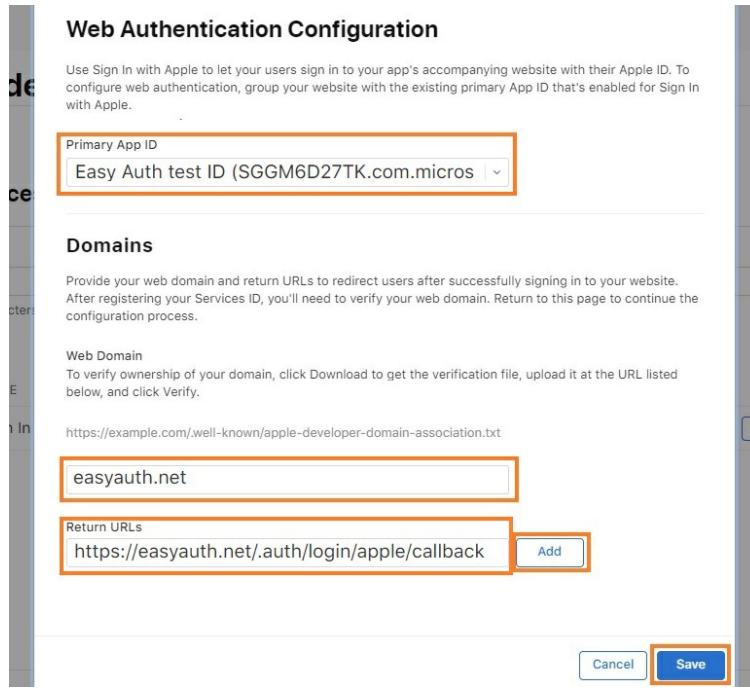
Identifier com.microsoft.easyauthtest.client

We recommend using a reverse-domain name style string (i.e., com.domainname.appname). It cannot contain an asterisk (\*).

ENABLED NAME  
 Sign in with Apple

[Configure](#)

9. On the pop-up window, set the Primary App ID to the App ID you created earlier. Specify your application's domain in the domain section. For the return URL, use the URL <app-url>/auth/login/apple/callback . For example, <https://contoso.azurewebsites.net/.auth/login/apple/callback>. Then select **Add** and **Save**.



10. Review the service registration information and select **Save**.

## Generate the client secret

Apple requires app developers to create and sign a JWT token as the client secret value. To generate this secret, first generate and download an elliptic curve private key from the Apple Developer portal. Then, use that key to [sign a JWT with a specific payload](#).

### Create and download the private key

1. On the **Keys** tab in the Apple Developer portal, choose **Create a key** or select the (+) button.
2. On the **Register a New Key** page give the key a name, check the box next to **Sign in with Apple** and select **Configure**.
3. On the **Configure Key** page, link the key to the primary app ID you created previously and select **Save**.
4. Finish creating the key by confirming the information and selecting **Continue** and then reviewing the information and selecting **Register**.
5. On the **Download Your Key** page, download the key. It will download as a **.p8** (PKCS#8) file - you'll use the file contents to sign your client secret JWT.

### Structure the client secret JWT

Apple requires the client secret be the base64-encoding of a JWT token. The decoded JWT token should have a payload structured like this example:

```
{
 "alg": "ES256",
 "kid": "URKEYID001",
}.{
 "sub": "com.yourcompany.app1",
 "nbf": 1560203207,
 "exp": 1560289607,
 "iss": "ABC123DEFG",
 "aud": "https://appleid.apple.com"
}.[Signature]
```

- **sub**: The Apple Client ID (also the service ID)
- **iss**: Your Apple Developer Team ID
- **aud**: Apple is receiving the token, so they're the audience
- **exp**: No more than six months after **nbf**

The base64-encoded version of the above payload looks like this:

```
eyJhbGciOiJFUzI1NiIsImtpZC161lVSS0VZSUwMDiFQ_eyJ2dwIi0Ijbj20ueW91cmNvbXBhbnkUYXBwMSIsIm5iZii6MTU2MDIwMzIwNywiZXhwIjoxNTYwMjg5NjA3LCJpc3MiOiJBQkMjNERUZHIIwi
pdzNg1seocoIcPNTmDKaz0-BHAZCsdeeTnlgFEzByTpmKFVEQDETGRkam5ieclUK7S9oOva4EK4jV4VmDrr-LGWW03TaAxAvy3_ZoKohvFFkVG
```

*Note: Apple doesn't accept client secret JWTs with an expiration date more than six months after the creation (or nbf) date. That means you'll need to rotate your client secret, at minimum, every six months.*

More information about generating and validating tokens can be found in [Apple's developer documentation](#).

### Sign the client secret JWT

You'll use the **.p8** file you downloaded previously to sign the client secret JWT. This file is a **PKCS#8** file that contains the private signing key in PEM format. There are many libraries that can create and sign the JWT for you.

There are different kinds of open-source libraries available online for creating and signing JWT tokens. For more information about generating JWT tokens, see [JSON Web Token \(JWT\)](#). For example, one way of generating the client secret is by importing the [Microsoft.IdentityModel.Tokens NuGet package](#) and running a small amount of C# code shown below.

```

using Microsoft.IdentityModel.Tokens;

public static string GetAppleClientSecret(string teamId, string clientId, string keyId, string p8key)
{
 string audience = "https://appleid.apple.com";

 string issuer = teamId;
 string subject = clientId;
 string kid = keyId;

 IList<Claim> claims = new List<Claim> {
 new Claim ("sub", subject)
 };

 CngKey cngKey = CngKey.Import(Convert.FromBase64String(p8key), CngKeyBlobFormat.Pkcs8PrivateBlob);

 SigningCredentials signingCred = new SigningCredentials(
 new ECDsaSecurityKey(new ECDsaCng(cngKey)),
 SecurityAlgorithms.EcdsaSha256
);

 JwtSecurityToken token = new JwtSecurityToken(
 issuer,
 audience,
 claims,
 DateTime.Now,
 DateTime.Now.AddDays(180),
 signingCred
);
 token.Header.Add("kid", kid);
 token.Header.Remove("typ");

 JwtSecurityTokenHandler tokenHandler = new JwtSecurityTokenHandler();

 return tokenHandler.WriteToken(token);
}

```

- **teamId:** Your Apple Developer Team ID
- **clientId:** The Apple Client ID (also the service ID)
- **p8key:** The PEM format key - you can obtain the key by opening the `.p8` file in a text editor, and copying everything between `-----BEGIN PRIVATE KEY-----` and `-----END PRIVATE KEY-----` without line breaks
- **keyId:** The ID of the downloaded key

This token returned is the client secret value you'll use to configure the Apple provider.

#### IMPORTANT

The client secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

Add the client secret as an [application setting](#) for the app, using a setting name of your choice. Make note of this name for later.

## Add provider information to your application

#### NOTE

The required configuration is in a new API format, currently only supported by [file-based configuration \(preview\)](#). You will need to follow the below steps using such a file.

This section will walk you through updating the configuration to include your new IDP. An example configuration follows.

1. Within the `identityProviders` object, add an `apple` object if one doesn't already exist.
2. Assign an object to that key with a `registration` object within it, and optionally a `login` object:

```

"apple" : {
 "registration" : {
 "clientId": "<client ID>",
 "clientSecretSettingName": "APP_SETTING_CONTAINING_APPLE_CLIENT_SECRET"
 },
 "login": {
 "scopes": []
 }
}

```

- a. Within the `registration` object, set the `clientId` to the client ID you collected.
- b. Within the `registration` object, set `clientSecretSettingName` to the name of the application setting where you stored the client secret.
- c. Within the `login` object, you may choose to set the `scopes` array to include a list of scopes used when authenticating with Apple, such as "name" and "email". If scopes are configured, they'll be explicitly requested on the consent screen when users sign in for the first time.

Once this configuration has been set, you're ready to use your Apple provider for authentication in your app.

A complete configuration might look like the following example (where the `APPLE_GENERATED_CLIENT_SECRET` setting points to an application setting containing a generated JWT):

```
{
 "platform": {
 "enabled": true
 },
 "globalValidation": {
 "redirectToProvider": "apple",
 "unauthenticatedClientAction": "RedirectToLoginPage"
 },
 "identityProviders": {
 "apple": {
 "registration": {
 "clientId": "com.contoso.example.client",
 "clientSecretSettingName": "APPLE_GENERATED_CLIENT_SECRET"
 },
 "login": {
 "scopes": []
 }
 }
 },
 "login": {
 "tokenStore": {
 "enabled": true
 }
 }
}
```

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#)

# Customize sign-in and sign-out in Azure App Service authentication

11/2/2021 • 6 minutes to read • [Edit Online](#)

This article shows you how to customize user sign-ins and sign-outs while using the built-in [authentication and authorization in App Service](#).

## Use multiple sign-in providers

The portal configuration doesn't offer a turn-key way to present multiple sign-in providers to your users (such as both Facebook and Twitter). However, it isn't difficult to add the functionality to your app. The steps are outlined as follows:

First, in the **Authentication / Authorization** page in the Azure portal, configure each of the identity provider you want to enable.

In **Action to take when request is not authenticated**, select **Allow Anonymous requests (no action)**.

In the sign-in page, or the navigation bar, or any other location of your app, add a sign-in link to each of the providers you enabled (`/auth/login/<provider>`). For example:

```
Log in with the Microsoft Identity Platform
Log in with Facebook
Log in with Google
Log in with Twitter
Log in with Apple
```

When the user clicks on one of the links, the respective sign-in page opens to sign in the user.

To redirect the user post-sign-in to a custom URL, use the `post_login_redirect_uri` query string parameter (not to be confused with the Redirect URI in your identity provider configuration). For example, to navigate the user to `/Home/Index` after sign-in, use the following HTML code:

```
<a href="/.auth/login/<provider>?post_login_redirect_uri=/Home/Index">Log in
```

## Client-directed sign-in

In a client-directed sign-in, the application signs in the user to the identity provider using a provider-specific SDK. The application code then submits the resulting authentication token to App Service for validation (see [Authentication flow](#)) using an HTTP POST request. The [Azure Mobile Apps SDKs](#) use this sign-in flow. This validation itself doesn't actually grant you access to the desired app resources, but a successful validation will give you a session token that you can use to access app resources.

To validate the provider token, App Service app must first be configured with the desired provider. At runtime, after you retrieve the authentication token from your provider, post the token to `/auth/login/<provider>` for validation. For example:

```

POST https://<appname>.azurewebsites.net/.auth/login/aad HTTP/1.1
Content-Type: application/json

{"id_token": "<token>", "access_token": "<token>"}

```

The token format varies slightly according to the provider. See the following table for details:

Provider Value	Required in Request Body	Comments
aad	{"access_token": "<access_token>"}	The <code>id_token</code> , <code>refresh_token</code> , and <code>expires_in</code> properties are optional.
microsoftaccount	{"access_token": "<access_token>"} or {"authentication_token": "<token>"}	<code>authentication_token</code> is preferred over <code>access_token</code> . The <code>expires_in</code> property is optional. When requesting the token from Live services, always request the <code>wl.basic</code> scope.
google	{"id_token": "<id_token>"}	The <code>authorization_code</code> property is optional. Providing an <code>authorization_code</code> value will add an access token and a refresh token to the token store. When specified, <code>authorization_code</code> can also optionally be accompanied by a <code>redirect_uri</code> property.
facebook	{"access_token": "<user_access_token>"}	Use a valid <a href="#">user access token</a> from Facebook.
twitter	{"access_token": "<access_token>", "access_token_secret": "<access_token_secret>"}	

If the provider token is validated successfully, the API returns with an `authenticationToken` in the response body, which is your session token.

```
{
 "authenticationToken": "...",
 "user": {
 "userId": "sid:..."
 }
}
```

Once you have this session token, you can access protected app resources by adding the `X-ZUMO-AUTH` header to your HTTP requests. For example:

```

GET https://<appname>.azurewebsites.net/api/products/1
X-ZUMO-AUTH: <authenticationToken_value>

```

## Sign out of a session

Users can initiate a sign-out by sending a `GET` request to the app's `/auth/logout` endpoint. The `GET` request does the following:

- Clears authentication cookies from the current session.
- Deletes the current user's tokens from the token store.
- For Azure Active Directory and Google, performs a server-side sign-out on the identity provider.

Here's a simple sign-out link in a webpage:

```
Sign out
```

By default, a successful sign-out redirects the client to the URL `/auth/logout/done`. You can change the post-sign-out redirect page by adding the `post_logout_redirect_uri` query parameter. For example:

```
GET /auth/logout?post_logout_redirect_uri=/index.html
```

It's recommended that you [encode](#) the value of `post_logout_redirect_uri`.

When using fully qualified URLs, the URL must be either hosted in the same domain or configured as an allowed external redirect URL for your app. In the following example, to redirect to `https://myexternalurl.com` that's not hosted in the same domain:

```
GET /auth/logout?post_logout_redirect_uri=https%3A%2F%2Fmyexternalurl.com
```

Run the following command in the [Azure Cloud Shell](#):

```
az webapp auth update --name <app_name> --resource-group <group_name> --allowed-external-redirect-urls "https://myexternalurl.com"
```

## Preserve URL fragments

After users sign in to your app, they usually want to be redirected to the same section of the same page, such as `/wiki/Main_Page#SectionZ`. However, because [URL fragments](#) (for example, `#SectionZ`) are never sent to the server, they are not preserved by default after the OAuth sign-in completes and redirects back to your app. Users then get a suboptimal experience when they need to navigate to the desired anchor again. This limitation applies to all server-side authentication solutions.

In App Service authentication, you can preserve URL fragments across the OAuth sign-in. To do this, set an app setting called `WEBSITE_AUTH_PRESERVE_URL_FRAGMENT` to `true`. You can do it in the [Azure portal](#), or simply run the following command in the [Azure Cloud Shell](#):

```
az webapp config appsettings set --name <app_name> --resource-group <group_name> --settings WEBSITE_AUTH_PRESERVE_URL_FRAGMENT="true"
```

## Limit the domain of sign-in accounts

Both Microsoft Account and Azure Active Directory lets you sign in from multiple domains. For example, Microsoft Account allows `outlook.com`, `live.com`, and `hotmail.com` accounts. Azure AD allows any number of custom domains for the sign-in accounts. However, you may want to accelerate your users straight to your own branded Azure AD sign-in page (such as `contoso.com`). To suggest the domain name of the sign-in accounts, follow these steps.

In <https://resources.azure.com>, navigate to `subscriptions > <subscription_name> resourceGroups > <resource_group_name> > providers > Microsoft.Web > sites > <app_name> > config > authsettings`.

Click **Edit**, modify the following property, and then click **Put**. Be sure to replace `<domain_name>` with the domain you want.

```
"additionalLoginParams": ["domain_hint=<domain_name>"]
```

This setting appends the `domain_hint` query string parameter to the login redirect URL.

#### IMPORTANT

It's possible for the client to remove the `domain_hint` parameter after receiving the redirect URL, and then login with a different domain. So while this function is convenient, it's not a security feature.

## Authorize or deny users

While App Service takes care of the simplest authorization case (i.e. reject unauthenticated requests), your app may require more fine-grained authorization behavior, such as limiting access to only a specific group of users. In certain cases, you need to write custom application code to allow or deny access to the signed-in user. In other cases, App Service or your identity provider may be able to help without requiring code changes.

- [Server level](#)
- [Identity provider level](#)
- [Application level](#)

### Server level (Windows apps only)

For any Windows app, you can define authorization behavior of the IIS web server, by editing the `Web.config` file. Linux apps don't use IIS and can't be configured through `Web.config`.

1. Navigate to <https://<app-name>.scm.azurewebsites.net/DebugConsole>
2. In the browser explorer of your App Service files, navigate to `site/wwwroot`. If a `Web.config` doesn't exist, create it by selecting **+** > **New File**.
3. Select the pencil for `Web.config` to edit it. Add the following configuration code and click **Save**. If `Web.config` already exists, just add the `<authorization>` element with everything in it. Add the accounts you want to allow in the `<allow>` element.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <system.web>
 <authorization>
 <allow users="user1@contoso.com,user2@contoso.com"/>
 <deny users="*"/>
 </authorization>
 </system.web>
</configuration>
```

### Identity provider level

The identity provider may provide certain turn-key authorization. For example:

- For [Azure App Service](#), you can [manage enterprise-level access](#) directly in Azure AD. For instructions, see [How to remove a user's access to an application](#).

- For [Google](#), Google API projects that belong to an [organization](#) can be configured to allow access only to users in your organization (see [Google's Setting up OAuth 2.0 support page](#)).

### Application level

If either of the other levels don't provide the authorization you need, or if your platform or identity provider isn't supported, you must write custom code to authorize users based on the [user claims](#).

## More resources

- [Tutorial: Authenticate and authorize users end-to-end](#)
- [Environment variables and app settings for authentication](#)

# Work with user identities in Azure App Service authentication

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to work with user identities when using the built-in [authentication and authorization in App Service](#).

## Access user claims in app code

For all language frameworks, App Service makes the claims in the incoming token (whether from an authenticated end user or a client application) available to your code by injecting them into the request headers. External requests aren't allowed to set these headers, so they are present only if set by App Service. Some example headers include:

- X-MS-CLIENT-PRINCIPAL-NAME
- X-MS-CLIENT-PRINCIPAL-ID

Code that is written in any language or framework can get the information that it needs from these headers.

### NOTE

Different language frameworks may present these headers to the app code in different formats, such as lowercase or title case.

For ASP.NET 4.6 apps, App Service populates `ClaimsPrincipal.Current` with the authenticated user's claims, so you can follow the standard .NET code pattern, including the `[Authorize]` attribute. Similarly, for PHP apps, App Service populates the `_SERVER['REMOTE_USER']` variable. For Java apps, the claims are [accessible from the Tomcat servlet](#).

For [Azure Functions](#), `ClaimsPrincipal.Current` is not populated for .NET code, but you can still find the user claims in the request headers, or get the `ClaimsPrincipal` object from the request context or even through a binding parameter. See [working with client identities in Azure Functions](#) for more information.

For .NET Core, [Microsoft.Identity.Web](#) supports populating the current user with App Service authentication. To learn more, you can read about it on the [Microsoft.Identity.Web wiki](#), or see it demonstrated in [this tutorial for a web app accessing Microsoft Graph](#).

## Access user claims using the API

If the [token store](#) is enabled for your app, you can also obtain additional details on the authenticated user by calling `/auth/me`. The Mobile Apps server SDKs provide helper methods to work with this data. For more information, see [How to use the Azure Mobile Apps Node.js SDK](#), and [Work with the .NET backend server SDK for Azure Mobile Apps](#).

## Next steps

[Tutorial: Authenticate and authorize users end-to-end](#)

# Work with OAuth tokens in Azure App Service authentication

11/2/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to work with OAuth tokens while using the built-in [authentication and authorization in App Service](#).

## Retrieve tokens in app code

From your server code, the provider-specific tokens are injected into the request header, so you can easily access them. The following table shows possible token header names:

Provider	Header Names
Azure Active Directory	X-MS-TOKEN-AAD-ID-TOKEN X-MS-TOKEN-AAD-ACCESS-TOKEN X-MS-TOKEN-AAD-EXPIRES-ON X-MS-TOKEN-AAD-REFRESH-TOKEN
Facebook Token	X-MS-TOKEN-FACEBOOK-ACCESS-TOKEN X-MS-TOKEN-FACEBOOK-EXPIRES-ON
Google	X-MS-TOKEN-GOOGLE-ID-TOKEN X-MS-TOKEN-GOOGLE-ACCESS-TOKEN X-MS-TOKEN-GOOGLE-EXPIRES-ON X-MS-TOKEN-GOOGLE-REFRESH-TOKEN
Twitter	X-MS-TOKEN-TWITTER-ACCESS-TOKEN X-MS-TOKEN-TWITTER-ACCESS-TOKEN-SECRET

### Note

Different language frameworks may present these headers to the app code in different formats, such as lowercase or title case.

From your client code (such as a mobile app or in-browser JavaScript), send an HTTP `GET` request to `/auth/me` ([token store](#) must be enabled). The returned JSON has the provider-specific tokens.

### Note

Access tokens are for accessing provider resources, so they are present only if you configure your provider with a client secret. To see how to get refresh tokens, see [Refresh access tokens](#).

## Refresh auth tokens

When your provider's access token (not the [session token](#)) expires, you need to reauthenticate the user before you use that token again. You can avoid token expiration by making a `GET` call to the `/auth/refresh` endpoint

of your application. When called, App Service automatically refreshes the access tokens in the [token store](#) for the authenticated user. Subsequent requests for tokens by your app code get the refreshed tokens. However, for token refresh to work, the token store must contain [refresh tokens](#) for your provider. The way to get refresh tokens are documented by each provider, but the following list is a brief summary:

- **Google:** Append an `access_type=offline` query string parameter to your `/.auth/login/google` API call. If using the Mobile Apps SDK, you can add the parameter to one of the `LogicAsync` overloads (see [Google Refresh Tokens](#)).
- **Facebook:** Doesn't provide refresh tokens. Long-lived tokens expire in 60 days (see [Facebook Expiration and Extension of Access Tokens](#)).
- **Twitter:** Access tokens don't expire (see [Twitter OAuth FAQ](#)).
- **Microsoft:** In <https://resources.azure.com>, do the following steps:
  1. At the top of the page, select **Read/Write**.
  2. In the left browser, navigate to `subscriptions > <subscription_name> > resourceGroups > <resource_group_name> > providers > Microsoft.Web > sites > <app_name> > config > authsettingsV2`.
  3. Click **Edit**.
  4. Modify the following property.

```
"identityProviders": {
 "azureActiveDirectory": {
 "login": {
 "loginParameters": ["scope=openid profile email offline_access"]
 }
 }
}
```

5. Click **Put**.

#### NOTE

The scope that gives you a refresh token is `offline_access`. See how it's used in [Tutorial: Authenticate and authorize users end-to-end in Azure App Service](#). The other scopes are requested by default by App Service already. For information on these default scopes, see [OpenID Connect Scopes](#).

Once your provider is configured, you can [find the refresh token and the expiration time for the access token](#) in the token store.

To refresh your access token at any time, just call `/.auth/refresh` in any language. The following snippet uses jQuery to refresh your access tokens from a JavaScript client.

```
function refreshTokens() {
 let refreshUrl = "/.auth/refresh";
 $.ajax(refreshUrl) .done(function() {
 console.log("Token refresh completed successfully.");
 }) .fail(function() {
 console.log("Token refresh failed. See application logs for details.");
 });
}
```

If a user revokes the permissions granted to your app, your call to `/.auth/me` may fail with a `403 Forbidden`

response. To diagnose errors, check your application logs for details.

## Extend session token expiration grace period

The authenticated session expires after 8 hours. After an authenticated session expires, there is a 72-hour grace period by default. Within this grace period, you're allowed to refresh the session token with App Service without reauthenticating the user. You can just call `/.auth/refresh` when your session token becomes invalid, and you don't need to track token expiration yourself. Once the 72-hour grace period is lapses, the user must sign in again to get a valid session token.

If 72 hours isn't enough time for you, you can extend this expiration window. Extending the expiration over a long period could have significant security implications (such as when an authentication token is leaked or stolen). So you should leave it at the default 72 hours or set the extension period to the smallest value.

To extend the default expiration window, run the following command in the [Cloud Shell](#).

```
az webapp auth update --resource-group <group_name> --name <app_name> --token-refresh-extension-hours <hours>
```

### NOTE

The grace period only applies to the App Service authenticated session, not the tokens from the identity providers. There is no grace period for the expired provider tokens.

## Next steps

[Tutorial: Authenticate and authorize users end-to-end](#)

# Manage the API and runtime versions of App Service authentication

11/2/2021 • 8 minutes to read • [Edit Online](#)

This article shows you how to customize the API and runtime versions of the built-in [authentication and authorization in App Service](#).

There are two versions of the management API for App Service authentication. The V2 version is required for the "Authentication" experience in the Azure portal. An app already using the V1 API can upgrade to the V2 version once a few changes have been made. Specifically, secret configuration must be moved to slot-sticky application settings. This can be done automatically from the "Authentication" section of the portal for your app.

## Update the configuration version

### WARNING

Migration to V2 will disable management of the App Service Authentication / Authorization feature for your application through some clients, such as its existing experience in the Azure portal, Azure CLI, and Azure PowerShell. This cannot be reversed.

The V2 API does not support creation or editing of Microsoft Account as a distinct provider as was done in V1. Rather, it leverages the converged [Microsoft Identity Platform](#) to sign-in users with both Azure AD and personal Microsoft accounts. When switching to the V2 API, the V1 Azure Active Directory configuration is used to configure the Microsoft Identity Platform provider. The V1 Microsoft Account provider will be carried forward in the migration process and continue to operate as normal, but it is recommended that you move to the newer Microsoft Identity Platform model. See [Support for Microsoft Account provider registrations](#) to learn more.

The automated migration process will move provider secrets into application settings and then convert the rest of the configuration into the new format. To use the automatic migration:

1. Navigate to your app in the portal and select the **Authentication** menu option.
2. If the app is configured using the V1 model, you will see an **Upgrade** button.
3. Review the description in the confirmation prompt. If you are ready to perform the migration, click **Upgrade** in the prompt.

### Manually managing the migration

The following steps will allow you to manually migrate the application to the V2 API if you do not wish to use the automatic version mentioned above.

#### Moving secrets to application settings

1. Get your existing configuration by using the V1 API:

```
az webapp auth show -g <group_name> -n <site_name>
```

In the resulting JSON payload, make note of the secret value used for each provider you have configured:

- AAD: `clientSecret`
- Google: `googleClientSecret`
- Facebook: `facebookAppSecret`

- Twitter: `twitterConsumerSecret`
- Microsoft Account: `microsoftAccountClientSecret`

#### IMPORTANT

The secret values are important security credentials and should be handled carefully. Do not share these values or persist them on a local machine.

2. Create slot-sticky application settings for each secret value. You may choose the name of each application setting. Its value should match what you obtained in the previous step or [reference a Key Vault secret](#) that you have created with that value.

To create the setting, you can use the Azure portal or run a variation of the following for each provider:

```
For Web Apps, Google example
az webapp config appsettings set -g <group_name> -n <site_name> --slot-settings
GOOGLE_PROVIDER_AUTHENTICATION_SECRET=<value_from_previous_step>

For Azure Functions, Twitter example
az functionapp config appsettings set -g <group_name> -n <site_name> --slot-settings
TWITTER_PROVIDER_AUTHENTICATION_SECRET=<value_from_previous_step>
```

#### NOTE

The application settings for this configuration should be marked as slot-sticky, meaning that they will not move between environments during a [slot swap operation](#). This is because your authentication configuration itself is tied to the environment.

3. Create a new JSON file named `authsettings.json`. Take the output that you received previously and remove each secret value from it. Write the remaining output to the file, making sure that no secret is included. In some cases, the configuration may have arrays containing empty strings. Make sure that `microsoftAccountOAuthScopes` does not, and if it does, switch that value to `null`.
4. Add a property to `authsettings.json` which points to the application setting name you created earlier for each provider:

- AAD: `clientSecretSettingName`
- Google: `googleClientSecretSettingName`
- Facebook: `facebookAppSecretSettingName`
- Twitter: `twitterConsumerSecretSettingName`
- Microsoft Account: `microsoftAccountClientSecretSettingName`

An example file after this operation might look similar to the following, in this case only configured for AAD:

```
{
 "id": "/subscriptions/00d563f8-5b89-4c6a-bcec-
c1b9f6d607e0/resourceGroups/myresourcegroup/providers/Microsoft.Web/sites/mywebapp/config/authsettings",
 "name": "authsettings",
 "type": "Microsoft.Web/sites/config",
 "location": "Central US",
 "properties": {
 "enabled": true,
 "runtimeVersion": "~1",
 "unauthenticatedClientAction": "AllowAnonymous",
 "tokenStoreEnabled": true,
 "allowedExternalRedirectUrls": null,
 "defaultProvider": "AzureActiveDirectory",
 "clientId": "3197c8ed-2470-480a-8fae-58c25558ac9b",
 "clientSecret": "",
 "clientSecretSettingName": "MICROSOFT_IDENTITY_AUTHENTICATION_SECRET",
 "clientSecretCertificateThumbprint": null,
 "issuer": "https://sts.windows.net/0b2ef922-672a-4707-9643-9a5726eecd524/",
 "allowedAudiences": [
 "https://mywebapp.azurewebsites.net"
],
 "additionalLoginParams": null,
 "isAadAutoProvisioned": true,
 "aadClaimsAuthorization": null,
 "googleClientId": null,
 "googleClientSecret": null,
 "googleClientSecretSettingName": null,
 "googleOAuthScopes": null,
 "facebookAppId": null,
 "facebookAppSecret": null,
 "facebookAppSecretSettingName": null,
 "facebookOAuthScopes": null,
 "gitHubClientId": null,
 "gitHubClientSecret": null,
 "gitHubClientSecretSettingName": null,
 "gitHubOAuthScopes": null,
 "twitterConsumerKey": null,
 "twitterConsumerSecret": null,
 "twitterConsumerSecretSettingName": null,
 "microsoftAccountClientId": null,
 "microsoftAccountClientSecret": null,
 "microsoftAccountClientSecretSettingName": null,
 "microsoftAccountOAuthScopes": null,
 "isAuthFromFile": "false"
 }
}
```

5. Submit this file as the new Authentication/Authorization configuration for your app:

```
az rest --method PUT --url
"/subscriptions/<subscription_id>/resourceGroups/<group_name>/providers/Microsoft.Web/sites/<site_name>/config/authsettings?api-version=2020-06-01" --body @./authsettings.json
```

6. Validate that your app is still operating as expected after this gesture.

7. Delete the file used in the previous steps.

You have now migrated the app to store identity provider secrets as application settings.

#### **Support for Microsoft Account provider registrations**

If your existing configuration contains a Microsoft Account provider and does not contain an Azure Active Directory provider, you can switch the configuration over to the Azure Active Directory provider and then perform the migration. To do this:

1. Go to [App registrations](#) in the Azure portal and find the registration associated with your Microsoft Account provider. It may be under the "Applications from personal account" heading.
2. Navigate to the "Authentication" page for the registration. Under "Redirect URLs" you should see an entry ending in `/auth/login/microsoftaccount/callback`. Copy this URI.
3. Add a new URI that matches the one you just copied, except instead have it end in `/auth/login/aad/callback`. This will allow the registration to be used by the App Service Authentication / Authorization configuration.
4. Navigate to the App Service Authentication / Authorization configuration for your app.
5. Collect the configuration for the Microsoft Account provider.
6. Configure the Azure Active Directory provider using the "Advanced" management mode, supplying the client ID and client secret values you collected in the previous step. For the Issuer URL, use Use `<authentication-endpoint>/<tenant-id>/v2.0`, and replace `<authentication-endpoint>` with the [authentication endpoint for your cloud environment](#) (e.g., "<https://login.microsoftonline.com>" for global Azure), also replacing `<tenant-id>` with your **Directory (tenant) ID**.
7. Once you have saved the configuration, test the login flow by navigating in your browser to the `/auth/login/aad` endpoint on your site and complete the sign-in flow.
8. At this point, you have successfully copied the configuration over, but the existing Microsoft Account provider configuration remains. Before you remove it, make sure that all parts of your app reference the Azure Active Directory provider through login links, etc. Verify that all parts of your app work as expected.
9. Once you have validated that things work against the AAD Azure Active Directory provider, you may remove the Microsoft Account provider configuration.

#### WARNING

It is possible to converge the two registrations by modifying the [supported account types](#) for the AAD app registration. However, this would force a new consent prompt for Microsoft Account users, and those users' identity claims may be different in structure, notably changing values since a new App ID is being used. This approach is not recommended unless thoroughly understood. You should instead wait for support for the two registrations in the V2 API surface.

#### Switching to V2

Once the above steps have been performed, navigate to the app in the Azure portal. Select the "Authentication (preview)" section.

Alternatively, you may make a PUT request against the `config/authsettingsv2` resource under the site resource. The schema for the payload is the same as captured in [File-based configuration](#).

## Pin your app to a specific authentication runtime version

When you enable Authentication / Authorization, platform middleware is injected into your HTTP request pipeline as described in the [feature overview](#). This platform middleware is periodically updated with new features and improvements as part of routine platform updates. By default, your web or function app will run on the latest version of this platform middleware. These automatic updates are always backwards compatible. However, in the rare event that this automatic update introduces a runtime issue for your web or function app, you can temporarily roll back to the previous middleware version. This article explains how to temporarily pin an app to a specific version of the authentication middleware.

#### Automatic and manual version updates

You can pin your app to a specific version of the platform middleware by setting a `runtimeVersion` setting for the app. Your app always runs on the latest version unless you choose to explicitly pin it back to a specific version. There will be a few versions supported at a time. If you pin to an invalid version that is no longer supported, your app will use the latest version instead. To always run the latest version, set `runtimeVersion` to `~1`.

## View and update the current runtime version

You can change the runtime version used by your app. The new runtime version should take effect after restarting the app.

### View the current runtime version

You can view the current version of the platform authentication middleware either using the Azure CLI or via one of the built-in version HTTP endpoints in your app.

#### From the Azure CLI

Using the Azure CLI, view the current middleware version with the [az webapp auth show](#) command.

```
az webapp auth show --name <my_app_name> \
--resource-group <my_resource_group>
```

In this code, replace `<my_app_name>` with the name of your app. Also replace `<my_resource_group>` with the name of the resource group for your app.

You will see the `runtimeVersion` field in the CLI output. It will resemble the following example output, which has been truncated for clarity:

```
{
 "additionalLoginParams": null,
 "allowedAudiences": null,
 ...
 "runtimeVersion": "1.3.2",
 ...
}
```

#### From the version endpoint

You can also hit `/auth/version` endpoint on an app also to view the current middleware version that the app is running on. It will resemble the following example output:

```
{
 "version": "1.3.2"
}
```

### Update the current runtime version

Using the Azure CLI, you can update the `runtimeVersion` setting in the app with the [az webapp auth update](#) command.

```
az webapp auth update --name <my_app_name> \
--resource-group <my_resource_group> \
--runtime-version <version>
```

Replace `<my_app_name>` with the name of your app. Also replace `<my_resource_group>` with the name of the resource group for your app. Also, replace `<version>` with a valid version of the 1.x runtime or `~1` for the latest version. See the [release notes on the different runtime versions](#) to help determine the version to pin to.

You can run this command from the [Azure Cloud Shell](#) by choosing Try it in the preceding code sample. You can also use the [Azure CLI locally](#) to execute this command after executing [az login](#) to sign in.

## Next steps

[Tutorial: Authenticate and authorize users end-to-end](#)

# File-based configuration in Azure App Service authentication

11/2/2021 • 3 minutes to read • [Edit Online](#)

With [App Service authentication](#), the authentication settings can be configured with a file. You may need to use file-based configuration to use certain preview capabilities of App Service authentication / authorization before they are exposed via [Azure Resource Manager APIs](#).

## IMPORTANT

Remember that your app payload, and therefore this file, may move between environments, as with [slots](#). It is likely you would want a different app registration pinned to each slot, and in these cases, you should continue to use the standard configuration method instead of using the configuration file.

## Enabling file-based configuration

1. Create a new JSON file for your configuration at the root of your project (deployed to `D:\home\site\wwwroot` in your web / function app). Fill in your desired configuration according to the [file-based configuration reference](#). If modifying an existing Azure Resource Manager configuration, make sure to translate the properties captured in the `authsettings` collection into your configuration file.
2. Modify the existing configuration, which is captured in the [Azure Resource Manager APIs](#) under `Microsoft.Web/sites/<siteName>/config/authsettingsV2`. To modify this, you can use an [Azure Resource Manager template](#) or a tool like [Azure Resource Explorer](#). Within the `authsettingsV2` collection, you will need to set two properties (and may remove others):
  - a. Set `platform.enabled` to "true"
  - b. Set `platform.configFilePath` to the name of the file (for example, "auth.json")

## NOTE

The format for `platform.configFilePath` varies between platforms. On Windows, both relative and absolute paths are supported. Relative is recommended. For Linux, only absolute paths are supported currently, so the value of the setting should be `"/home/site/wwwroot/auth.json"` or similar.

Once you have made this configuration update, the contents of the file will be used to define the behavior of App Service Authentication / Authorization for that site. If you ever wish to return to Azure Resource Manager configuration, you can do so by removing changing the setting `platform.configFilePath` to null.

## Configuration file reference

Any secrets that will be referenced from your configuration file must be stored as [application settings](#). You may name the settings anything you wish. Just make sure that the references from the configuration file uses the same keys.

The following exhausts possible configuration options within the file:

```
{
 "platform": {
```

```

 "enabled": <true|false>
 },
 "globalValidation": {
 "unauthenticatedClientAction": "RedirectToLoginPage|AllowAnonymous|RejectWith401|RejectWith404",
 "redirectToProvider": "<default provider alias>",
 "excludedPaths": [
 "/path1",
 "/path2"
]
 },
 "httpSettings": {
 "requireHttps": <true|false>,
 "routes": {
 "apiPrefix": "<api prefix>"
 },
 "forwardProxy": {
 "convention": "NoProxy|Standard|Custom",
 "customHostHeaderName": "<host header value>",
 "customProtoHeaderName": "<proto header value>"
 }
 },
 "login": {
 "routes": {
 "logoutEndpoint": "<logout endpoint>"
 },
 "tokenStore": {
 "enabled": <true|false>,
 "tokenRefreshExtensionHours": "<double>",
 "fileSystem": {
 "directory": "<directory to store the tokens in if using a file system token store
(default)>"
 },
 "azureBlobStorage": {
 "sasUrlSettingName": "<app setting name containing the sas url for the Azure Blob Storage if
opting to use that for a token store>"
 }
 },
 "preserveUrlFragmentsForLogins": <true|false>,
 "allowedExternalRedirectUrls": [
 "https://uri1.azurewebsites.net/",
 "https://uri2.azurewebsites.net/",
 "url_scheme_of_your_app://easyauth.callback"
],
 "cookieExpiration": {
 "convention": "FixedTime|IdentityDerived",
 "timeToExpiration": "<timespan>"
 },
 "nonce": {
 "validateNonce": <true|false>,
 "nonceExpirationInterval": "<timespan>"
 }
 },
 "identityProviders": {
 "azureActiveDirectory": {
 "enabled": <true|false>,
 "registration": {
 "openIdIssuer": "<issuer url>",
 "clientId": "<app id>",
 "clientSecretSettingName": "APP_SETTING_CONTAINING_AAD_SECRET",
 },
 "login": {
 "loginParameters": [
 "paramName1=value1",
 "paramName2=value2"
]
 },
 "validation": {
 "allowedAudiences": [
 "audience1",

```

```
 "audience2"
]
}
},
"facebook": {
 "enabled": <true|false>,
 "registration": {
 "appId": "<app id>",
 "appSecretSettingName": "APP_SETTING_CONTAINING_FACEBOOK_SECRET"
 },
 "graphApiVersion": "v3.3",
 "login": {
 "scopes": [
 "public_profile",
 "email"
]
 },
},
"gitHub": {
 "enabled": <true|false>,
 "registration": {
 "clientId": "<client id>",
 "clientSecretSettingName": "APP_SETTING_CONTAINING_GITHUB_SECRET"
 },
 "login": {
 "scopes": [
 "profile",
 "email"
]
 }
},
"google": {
 "enabled": true,
 "registration": {
 "clientId": "<client id>",
 "clientSecretSettingName": "APP_SETTING_CONTAINING_GOOGLE_SECRET"
 },
 "login": {
 "scopes": [
 "profile",
 "email"
]
 },
 "validation": {
 "allowedAudiences": [
 "audience1",
 "audience2"
]
 }
},
"twitter": {
 "enabled": <true|false>,
 "registration": {
 "consumerKey": "<consumer key>",
 "consumerSecretSettingName": "APP_SETTING_CONTAINING_TWITTER_CONSUMER_SECRET"
 }
},
"apple": {
 "enabled": <true|false>,
 "registration": {
 "clientId": "<client id>",
 "clientSecretSettingName": "APP_SETTING_CONTAINING_APPLE_SECRET"
 },
 "login": {
 "scopes": [
 "profile",
 "email"
]
 }
}
```

```

 },
 "openIdConnectProviders": {
 "<providerName>": {
 "enabled": <true|false>,
 "registration": {
 "clientId": "<client id>",
 "clientCredential": {
 "clientSecretSettingName": "<name of app setting containing client secret>"
 },
 "openIdConnectConfiguration": {
 "authorizationEndpoint": "<url specifying authorization endpoint>",
 "tokenEndpoint": "<url specifying token endpoint>",
 "issuer": "<url specifying issuer>",
 "certificationUri": "<url specifying jwks endpoint>",
 "wellKnownOpenIdConfiguration": "<url specifying .well-known/open-id-configuration
endpoint - if this property is set, the other properties of this object are ignored, and
authorizationEndpoint, tokenEndpoint, issuer, and certificationUri are set to the corresponding values
listed at this endpoint>"
 }
 },
 "login": {
 "nameClaimType": "<name of claim containing name>",
 "scopes": [
 "openid",
 "profile",
 "email"
],
 "loginParameterNames": [
 "paramName1=value1",
 "paramName2=value2"
],
 ...
 }
 },
 //...
 }
}

```

## More resources

- [Tutorial: Authenticate and authorize users end-to-end](#)
- [Environment variables and app settings for authentication](#)

# Add a TLS/SSL certificate in Azure App Service

11/2/2021 • 19 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This article shows you how to create, upload, or import a private certificate or a public certificate into App Service.

Once the certificate is added to your App Service app or [function app](#), you can [secure a custom DNS name with it](#) or [use it in your application code](#).

## NOTE

A certificate uploaded into an app is stored in a deployment unit that is bound to the app service plan's resource group and region combination (internally called a *webspace*). This makes the certificate accessible to other apps in the same resource group and region combination.

The following table lists the options you have for adding certificates in App Service:

OPTION	DESCRIPTION
Create a free App Service managed certificate	A private certificate that's free of charge and easy to use if you just need to secure your <a href="#">custom domain</a> in App Service.
Purchase an App Service certificate	A private certificate that's managed by Azure. It combines the simplicity of automated certificate management and the flexibility of renewal and export options.
Import a certificate from Key Vault	Useful if you use <a href="#">Azure Key Vault</a> to manage your <a href="#">PKCS12 certificates</a> . See <a href="#">Private certificate requirements</a> .
Upload a private certificate	If you already have a private certificate from a third-party provider, you can upload it. See <a href="#">Private certificate requirements</a> .
Upload a public certificate	Public certificates are not used to secure custom domains, but you can load them into your code if you need them to access remote resources.

## Prerequisites

- [Create an App Service app](#).
- For a private certificate, make sure that it satisfies all [requirements from App Service](#).
- **Free certificate only:**
  - Map the domain you want a certificate for to App Service. For information, see [Tutorial: Map an existing custom DNS name to Azure App Service](#).
  - For a root domain (like contoso.com), make sure your app doesn't have any [IP restrictions](#) configured. Both certificate creation and its periodic renewal for a root domain depends on your app being reachable from the internet.

## Private certificate requirements

The [free App Service managed certificate](#) and the [App Service certificate](#) already satisfy the requirements of App Service. If you choose to upload or import a private certificate to App Service, your certificate must meet the following requirements:

- Exported as a [password-protected PFX file](#), encrypted using triple DES.
- Contains private key at least 2048 bits long
- Contains all intermediate certificates and the root certificate in the certificate chain.

To secure a custom domain in a TLS binding, the certificate has additional requirements:

- Contains an [Extended Key Usage](#) for server authentication (OID = 1.3.6.1.5.5.7.3.1)
- Signed by a trusted certificate authority

#### NOTE

Elliptic Curve Cryptography (ECC) certificates can work with App Service but are not covered by this article. Work with your certificate authority on the exact steps to create ECC certificates.

## Prepare your web app

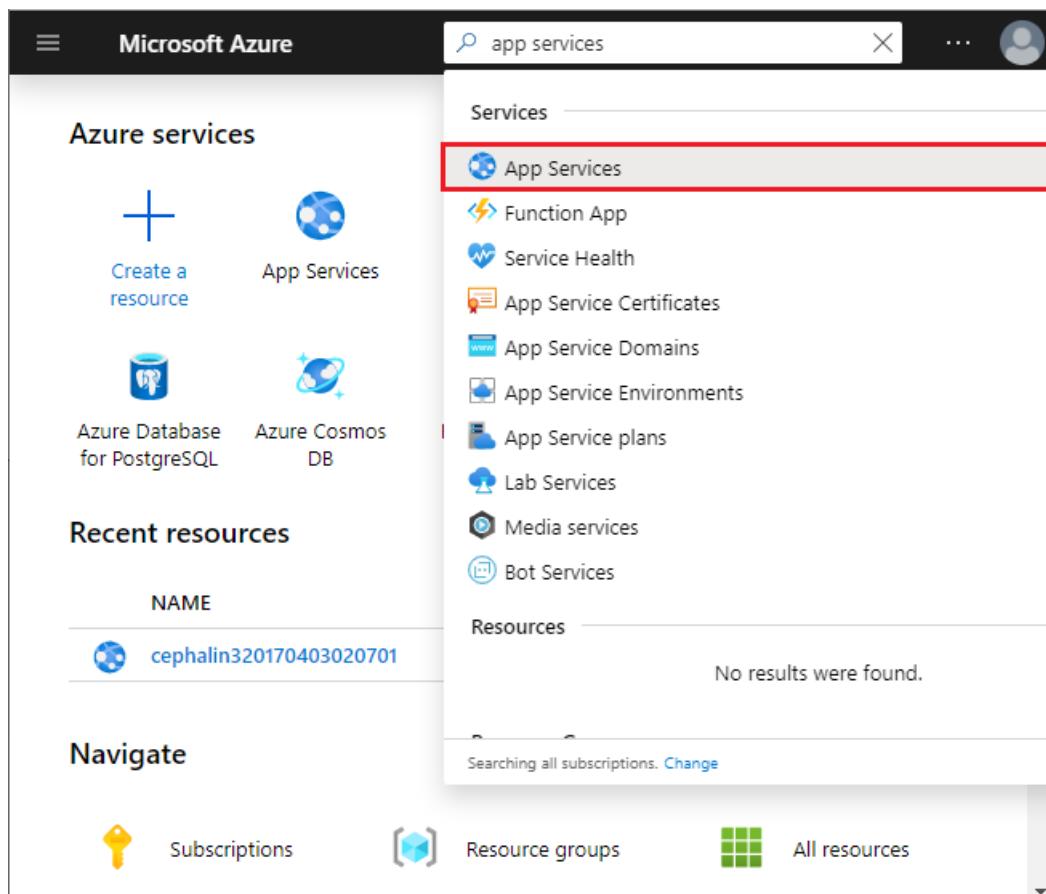
To create custom TLS/SSL bindings or enable client certificates for your App Service app, your [App Service plan](#) must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

### Sign in to Azure

Open the [Azure portal](#).

### Navigate to your web app

Search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "app services". Below the search bar, on the left, is a sidebar titled "Azure services" with icons for "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". On the right, the main content area shows a list of services under the heading "Services". The "App Services" item is highlighted with a red box. Other items in the list include "Function App", "Service Health", "App Service Certificates", "App Service Domains", "App Service Environments", "App Service plans", "Lab Services", "Media services", and "Bot Services". Below the service list is a section titled "Resources" with the message "No results were found.". At the bottom of the page, there are links for "Subscriptions", "Resource groups", and "All resources".

On the App Services page, select the name of your web app.

The screenshot shows the Azure App Services management interface. At the top, there's a navigation bar with 'Home > App Services'. Below it, a header bar includes 'App Services' (with a Microsoft logo), 'Documentation', and various action buttons like '+ Add', 'Edit columns', 'Refresh', 'Assign tags', 'Start', 'Restart', and 'More'. A message 'Subscriptions: All 2 selected - Don't see a subscription? Open Directory + Subscription settings' is displayed. Below this are several filter buttons: 'Filter by na...', 'All subsc... ▾', 'All resou... ▾', 'All locati... ▾', 'All tags ▾', and 'No group... ▾'. A table lists '6 items':

Name	Status	App Type	App Service
cephalin320170403020701	Running	Web App	test-sku
denniseastusbot	Running	Web App	z76-z763
myFirstAzureWebApp20190...	Running	Web App	ServicePl...
WebApplicationASPDotNET...	Running	Web App	ServicePl...

You have landed on the management page of your web app.

#### Check the pricing tier

In the left-hand navigation of your web app page, scroll to the **Settings** section and select **Scale up (App Service plan)**.

The screenshot shows the left sidebar of a web app's settings page. At the top is the app name 'cephalin320170403020701' and an 'App Service' badge. Below is a search bar. The sidebar has a 'Deployment Center' link and a 'Settings' section with the following options:

- Configuration
- Container settings
- Authentication / Authorization
- Application Insights
- Identity
- Backups
- Custom domains
- TLS/SSL settings
- Networking

The 'Scale up (App Service plan)' option is at the bottom of the sidebar and is highlighted with a red box.

Check to make sure that your web app is not in the F1 or D1 tier. Your web app's current tier is highlighted by a dark blue box.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

### Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:

#### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...

#### Memory

Memory available to run applications

Custom SSL is not supported in the F1 or D1 tier. If you need to scale up, follow the steps in the next section. Otherwise, close the Scale up page and skip the [Scale up your App Service plan](#) section.

#### Scale up your App Service plan

Select any of the non-free tiers (B1, B2, B3, or any tier in the Production category). For additional options, click See additional options.

Click **Apply**.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

## Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

## Included features

Every app hosted on this App Service plan will have access to these features:



### Custom domains / SSL

Configure and purchase custom domains with SNI SSL bindings



### Manual scale

Up to 3 instances. Subject to availability.

## Included hardware

Every instance of your App Service plan will include the following hardware configuration:



### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...



### Memory

Memory per instance available to run applications deployed and running in...



### Storage

10 GB disk storage shared by all apps deployed in the App Service plan.

**Apply**

When you see the following notification, the scale operation is complete.



Updating App Service Plan

3:14 PM

The plan 'Default1' was updated successfully!

## Create a free managed certificate

### NOTE

Before creating a free managed certificate, make sure you have [fulfilled the prerequisites](#) for your app.

The free App Service managed certificate is a turn-key solution for securing your custom DNS name in App Service. It's a TLS/SSL server certificate that's fully managed by App Service and renewed continuously and automatically in six-month increments, 45 days before expiration. You create the certificate and bind it to a custom domain, and let App Service do the rest.

The free certificate comes with the following limitations:

- Does not support wildcard certificates.
- Does not support usage as a client certificate by certificate thumbprint (removal of certificate thumbprint is planned).
- Is not exportable.
- Is not supported on App Service not publicly accessible.
- Is not supported on App Service Environment (ASE).
- Is not supported with root domains that are integrated with Traffic Manager.
- If a certificate is for a CNAME-mapped domain, the CNAME must be mapped directly to `<app-name>.azurewebsites.net`.

#### NOTE

The free certificate is issued by DigiCert. For some domains, you must explicitly allow DigiCert as a certificate issuer by creating a [CAA domain record](#) with the value: `0 issue digicert.com`.

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, select **TLS/SSL settings > Private Key Certificates (.pfx) > Create App Service Managed Certificate**.

The screenshot shows the Azure portal interface for managing certificates. On the left, there's a sidebar with various settings like Configuration, Authentication / Authorization, Application Insights, Identity, Backups, Custom domains, and TLS/SSL settings. The 'TLS/SSL settings' option is highlighted with a red box. The main area has tabs for 'Bindings', 'Private Key Certificates (.pfx)', and 'Public Key Certificates (.cer)'. The 'Private Key Certificates (.pfx)' tab is selected and also has a red box around it. Below the tabs, there's a sub-section for 'Private Key Certificate' with a 'PFX' icon. A descriptive text explains that private key certificates (.pfx) can be used for TLS/SSL bindings and loaded to the certificate store. It includes links for 'Import App Service Certificate', 'Upload Certificate', 'Import Key Vault Certificate', and 'Create App Service Managed Certificate'. The 'Create App Service Managed Certificate' button is specifically highlighted with a red box. At the bottom, there's a table for 'Private Key Certificates' with columns for Status Filter (All, Healthy, Warning, Expired), Health St..., Hostname, Expiration, and Thumbprint. A message states 'No private key certificates available for app.'

Select the custom domain to create a free certificate for and select **Create**. You can create only one certificate for each supported custom domain.

When the operation completes, you see the certificate in the **Private Key Certificates** list.

## Private Key Certificates

### Status Filter

All    Healthy    Warning    Expired

Health Status	Hostname	Expiration	Thumbprint	...
Healthy	www.contoso.com	4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

### IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

## Import an App Service Certificate

If you purchase an App Service Certificate from Azure, Azure manages the following tasks:

- Takes care of the purchase process from GoDaddy.
- Performs domain verification of the certificate.
- Maintains the certificate in [Azure Key Vault](#).
- Manages certificate renewal (see [Renew certificate](#)).
- Synchronize the certificate automatically with the imported copies in App Service apps.

To purchase an App Service certificate, go to [Start certificate order](#).

If you already have a working App Service certificate, you can:

- [Import the certificate into App Service](#).
- [Manage the certificate](#), such as renew, rekey, and export it.

### NOTE

App Service Certificates are not supported in Azure National Clouds at this time.

### Start certificate order

Start an App Service certificate order in the [App Service Certificate create page](#).

Use the following table to help you configure the certificate. When finished, click **Create**.

SETTING	DESCRIPTION
Name	A friendly name for your App Service certificate.
Naked Domain Host Name	Specify the root domain here. The issued certificate secures <i>both</i> the root domain and the <code>www</code> subdomain. In the issued certificate, the Common Name field contains the root domain, and the Subject Alternative Name field contains the <code>www</code> domain. To secure any subdomain only, specify the fully qualified domain name of the subdomain here (for example, <code>mysubdomain.contoso.com</code> ).
Subscription	The subscription that will contain the certificate.
Resource group	The resource group that will contain the certificate. You can use a new resource group or select the same resource group as your App Service app, for example.
Certificate SKU	Determines the type of certificate to create, whether a standard certificate or a <a href="#">wildcard certificate</a> .
Legal Terms	Click to confirm that you agree with the legal terms. The certificates are obtained from GoDaddy.

#### NOTE

App Service Certificates purchased from Azure are issued by GoDaddy. For some domains, you must explicitly allow GoDaddy as a certificate issuer by creating a [CAA domain record](#) with the value: `0 issue godaddy.com`

#### Store in Azure Key Vault

Once the certificate purchase process is complete, there are few more steps you need to complete before you

can start using this certificate.

Select the certificate in the [App Service Certificates](#) page, then click **Certificate Configuration > Step 1: Store**.

The screenshot shows the 'Certificate Configuration' page for an app service. On the left, a sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Settings, and Certificate Configuration (which is selected and highlighted with a red box). The main area displays three steps: Step 1: Store (highlighted), Step 2: Verify, and Step 3: Assign. Each step has an icon and a brief description. Step 1: Store has a key icon and says 'Import certificate into Keyvault for secure administration.' Step 2: Verify has a globe icon and says 'Verify certificate domain ownership.' Step 3: Assign has a cloud icon and says 'Certificate ready to use in App Service.'

[Key Vault](#) is an Azure service that helps safeguard cryptographic keys and secrets used by cloud applications and services. It's the storage of choice for App Service certificates.

In the [Key Vault Status](#) page, click **Key Vault Repository** to create a new vault or choose an existing vault. If you choose to create a new vault, use the following table to help you configure the vault and click **Create**. Create the new Key Vault inside the same subscription and resource group as your App Service app.

SETTING	DESCRIPTION
Name	A unique name that consists for alphanumeric characters and dashes.
Resource group	As a recommendation, select the same resource group as your App Service certificate.
Location	Select the same location as your App Service app.
Pricing tier	For information, see <a href="#">Azure Key Vault pricing details</a> .
Access policies	Defines the applications and the allowed access to the vault resources. You can configure it later, following the steps at <a href="#">Assign a Key Vault access policy</a> .
Virtual Network Access	Restrict vault access to certain Azure virtual networks. You can configure it later, following the steps at <a href="#">Configure Azure Key Vault Firewalls and Virtual Networks</a>

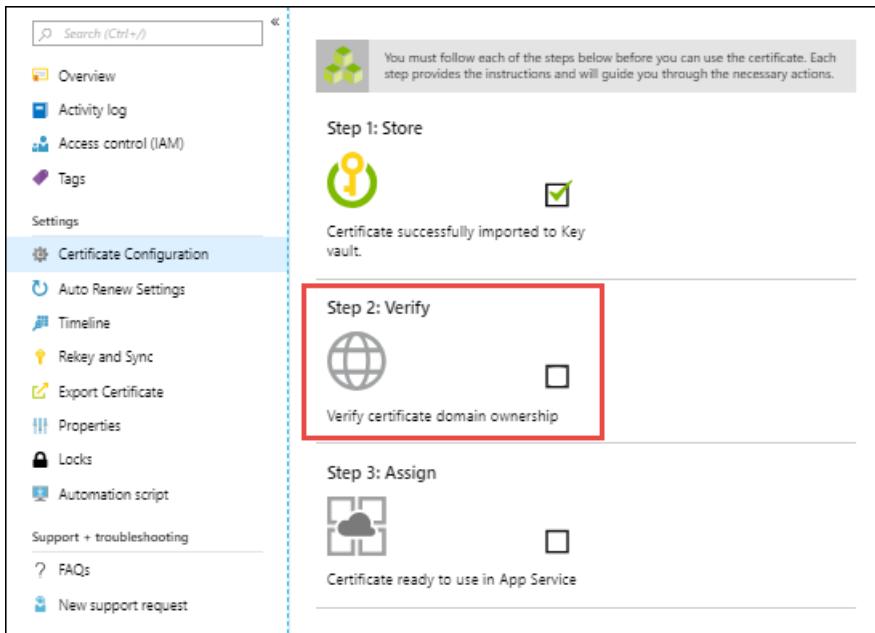
Once you've selected the vault, close the **Key Vault Repository** page. The **Step 1: Store** option should show a green check mark for success. Keep the page open for the next step.

#### NOTE

Currently, App Service Certificate only supports Key Vault access policy but not RBAC model.

## Verify domain ownership

From the same **Certificate Configuration** page you used in the last step, click **Step 2: Verify**.



Select **App Service Verification**. Since you already mapped the domain to your web app (see [Prerequisites](#)), it's already verified. Just click **Verify** to finish this step. Click the **Refresh** button until the message **Certificate is Domain Verified** appears.

#### NOTE

Four types of domain verification methods are supported:

- **App Service** - The most convenient option when the domain is already mapped to an App Service app in the same subscription. It takes advantage of the fact that the App Service app has already verified the domain ownership.
- **Domain** - Verify an [App Service domain that you purchased from Azure](#). Azure automatically adds the verification TXT record for you and completes the process.
- **Mail** - Verify the domain by sending an email to the domain administrator. Instructions are provided when you select the option.
- **Manual** - Verify the domain using either an HTML page (**Standard** certificate only) or a DNS TXT record. Instructions are provided when you select the option. The HTML page option doesn't work for web apps with "Https Only" enabled.

## Import certificate into App Service

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, select **TLS/SSL settings > Private Key Certificates (.pfx) > Import App Service Certificate**.

Private key certificates (.pfx) can be used for TLS/SSL bindings and can be loaded to the certificate store for your app to consume. To understand how to load the certificates for your app to consume click on the learn more link. Uploaded certificates are not available for manual download from the Azure Management Portal, they can only be used by your app hosted on App Service after the required App Settings are set properly or used for TLS/SSL. [Learn more](#)

**Private Key Certificate**

**Import Options:**

- Import App Service Certificate
- Upload Certificate
- Import Key Vault Certificate
- Create App Service Managed Certificate

**Private Key Certificates**

Status Filter: All (selected), Healthy, Warning, Expired

Health St...	Hostname	Expiration	Thumbprint	Actions
No private key certificates available for app.				

Select the certificate that you just purchased and select OK.

When the operation completes, you see the certificate in the **Private Key Certificates** list.

Health Status	Hostname	Expiration	Thumbprint	Actions
Healthy	contoso.com,www.contoso.com	10/10/2020	6A3BCCA5CC4B0158F0A097CE9F39...	...

### IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

## Import a certificate from Key Vault

If you use Azure Key Vault to manage your certificates, you can import a PKCS12 certificate from Key Vault into App Service as long as it [satisfies the requirements](#).

### Authorize App Service to read from the vault

By default, the App Service resource provider doesn't have access to the Key Vault. In order to use a Key Vault for a certificate deployment, you need to [authorize the resource provider read access to the KeyVault](#).

`abfa0a7c-a6b6-4736-8310-585508787cd` is the resource provider service principal name for App Service, and it's the same for all Azure subscriptions. For Azure Government cloud environment, use

`6a02c803-daf4-4136-b4c3-5a6f318b4714` instead as the resource provider service principal name.

### NOTE

Currently, Key Vault Certificate only supports Key Vault access policy but not RBAC model.

## Import a certificate from your vault to your app

In the [Azure portal](#), from the left menu, select App Services > <app-name>.

From the left navigation of your app, select **TLS/SSL settings** > **Private Key Certificates (.pfx)** > **Import**

## Key Vault Certificate.

Use the following table to help you select the certificate.

SETTING	DESCRIPTION
Subscription	The subscription that the Key Vault belongs to.
Key Vault	The vault with the certificate you want to import.
Certificate	Select from the list of PKCS12 certificates in the vault. All PKCS12 certificates in the vault are listed with their thumbprints, but not all are supported in App Service.

When the operation completes, you see the certificate in the **Private Key Certificates** list. If the import fails with an error, the certificate doesn't meet the [requirements for App Service](#).

### NOTE

If you update your certificate in Key Vault with a new certificate, App Service automatically syncs your certificate within 24 hours.

### IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

## Upload a private certificate

Once you obtain a certificate from your certificate provider, follow the steps in this section to make it ready for

App Service.

## Merge intermediate certificates

If your certificate authority gives you multiple certificates in the certificate chain, you need to merge the certificates in order.

To do this, open each certificate you received in a text editor.

Create a file for the merged certificate, called *mergedcertificate.crt*. In a text editor, copy the content of each certificate into this file. The order of your certificates should follow the order in the certificate chain, beginning with your certificate and ending with the root certificate. It looks like the following example:

```
-----BEGIN CERTIFICATE-----
<your entire Base64 encoded SSL certificate>
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
<The entire Base64 encoded intermediate certificate 1>
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
<The entire Base64 encoded intermediate certificate 2>
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
<The entire Base64 encoded root certificate>
-----END CERTIFICATE-----
```

## Export certificate to PFX

Export your merged TLS/SSL certificate with the private key that your certificate request was generated with.

If you generated your certificate request using OpenSSL, then you have created a private key file. To export your certificate to PFX, run the following command. Replace the placeholders <private-key-file> and <merged-certificate-file> with the paths to your private key and your merged certificate file.

```
openssl pkcs12 -export -out myserver.pfx -inkey <private-key-file> -in <merged-certificate-file>
```

When prompted, define an export password. You'll use this password when uploading your TLS/SSL certificate to App Service later.

If you used IIS or *Certreq.exe* to generate your certificate request, install the certificate to your local machine, and then [export the certificate to PFX](#).

## Upload certificate to App Service

You're now ready upload the certificate to App Service.

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, select **TLS/SSL settings > Private Key Certificates (.pfx) > Upload Certificate**.

Private key certificates (.pfx) can be used for TLS/SSL bindings and can be loaded to the certificate store for your app to consume. To understand how to load the certificates for your app to consume click on the learn more link. Uploaded certificates are not available for manual download from the Azure Management Portal, they can only be used by your app hosted on App Service after the required App Settings are set properly or used for TLS/SSL. [Learn more](#)

**Private Key Certificates**

Health St...	Hostname	Expiration	Thumbprint
Healthy	www.contoso.com	4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...

In **PFX Certificate File**, select your PFX file. In **Certificate password**, type the password that you created when you exported the PFX file. When finished, click **Upload**.

When the operation completes, you see the certificate in the **Private Key Certificates** list.

**Private Key Certificates**

Status Filter

All    Healthy    Warning    Expired

Health Status	Hostname	Expiration	Thumbprint
Healthy	www.contoso.com	4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...

### IMPORTANT

To secure a custom domain with this certificate, you still need to create a certificate binding. Follow the steps in [Create binding](#).

## Upload a public certificate

Public certificates are supported in the **.cer** format.

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, click **TLS/SSL settings > Public Certificates (.cer) > Upload Public Key Certificate**.

In **Name**, type a name for the certificate. In **CER Certificate file**, select your CER file.

Click **Upload**.

**Add Public Key Certificate (.cer)**

**Upload Public Key Certificate**

Upload a public key certificate (.cer) to be consumed in your app runtime. Note: Public key certificates cannot be used to configure TLS/SSL Bindings. [Learn more](#)

\* Name  
Contoso ✓

\* CER Certificate file  
"somecert.cer"

**Upload**

Once the certificate is uploaded, copy the certificate thumbprint and see [Make the certificate accessible](#).

## Renew an expiring certificate

Before a certificate expires, you should add the renewed certificate into App Service and update any [TLS/SSL binding](#). The process depends on the certificate type. For example, a [certificate imported from Key Vault](#), including an [App Service certificate](#), automatically syncs to App Service every 24 hours and updates the TLS/SSL binding when you renew the certificate. For an [uploaded certificate](#), there's no automatic binding update. See one of the following sections depending on your scenario:

- [Renew an uploaded certificate](#)
- [Renew an App Service certificate](#)
- [Renew a certificate imported from Key Vault](#)

### Renew an uploaded certificate

To replace an expiring certificate, how you update the certificate binding with the new certificate can adversely affect user experience. For example, your inbound IP address can change when you delete a binding, even if that binding is IP-based. This is especially important when you renew a certificate that's already in an IP-based binding. To avoid a change in your app's IP address, and to avoid downtime for your app due to HTTPS errors, follow these steps in order:

1. [Upload the new certificate](#).
2. [Bind the new certificate to the same custom domain](#) without deleting the existing (expiring) certificate. This action replaces the binding instead of removing the existing certificate binding.
3. Delete the existing certificate.

### Renew an App Service certificate

#### NOTE

Starting September 23 2021, App Service certificates will require domain validation every 395 days. This is due to a new guideline enforced by the CA/Browser Forum that Certificate Authorities must comply with.

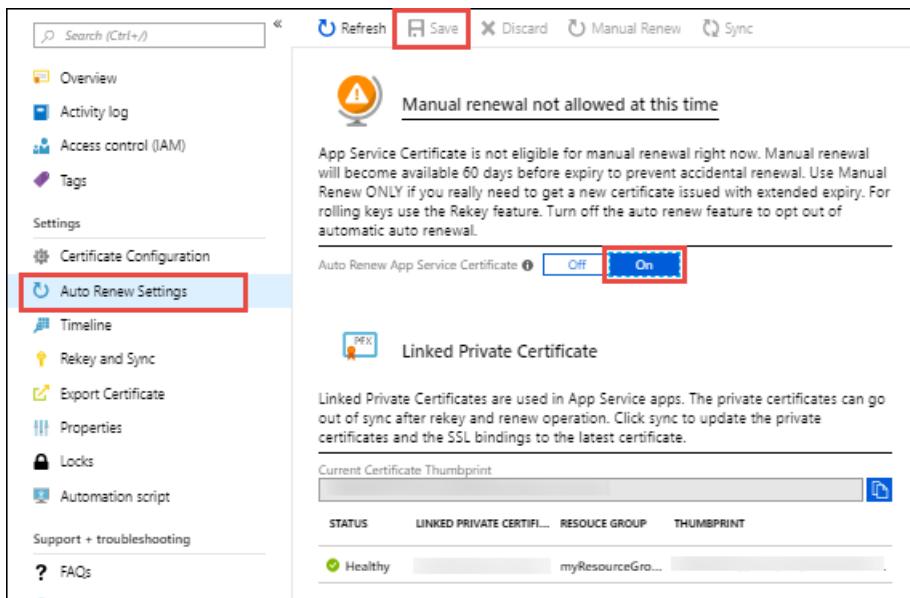
Unlike App Service Managed Certificate, domain re-validation for App Service Certificate will NOT be automated. Refer to [verify domain ownership](#) for more information on how to verify your App Service certificate.

#### NOTE

The renewal process requires that [the well-known service principal for App Service has the required permissions on your key vault](#). This permission is configured for you when you import an App Service Certificate through the portal, and should not be removed from your key vault.

To toggle the automatic renewal setting of your App Service certificate at any time, select the certificate in the [App Service Certificates](#) page, then click **Auto Renew Settings** in the left navigation. By default, App Service Certificates have a one-year validity period.

Select **On** or **Off** and click **Save**. Certificates can start automatically renewing 31 days before expiration if you have automatic renewal turned on.



To manually renew the certificate instead, click **Manual Renew**. You can request to manually renew your certificate 60 days before expiration.

Once the renew operation is complete, click **Sync**. The sync operation automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

#### NOTE

If you don't click **Sync**, App Service automatically syncs your certificate within 24 hours.

### Renew a certificate imported from Key Vault

To renew a certificate you imported into App Service from Key Vault, see [Renew your Azure Key Vault certificate](#).

Once the certificate is renewed in your key vault, App Service automatically syncs the new certificate and updates any applicable TLS/SSL binding within 24 hours. To sync manually:

1. Go to your app's **TLS/SSL settings** page.

2. Select the imported certificate under **Private Key Certificates**.

3. Click **Sync**.

## Manage App Service certificates

This section shows you how to manage an [App Service certificate you purchased](#).

- [Rekey certificate](#)
- [Export certificate](#)
- [Delete certificate](#)

Also, see [Renew an App Service certificate](#)

### Rekey certificate

If you think your certificate's private key is compromised, you can rekey your certificate. Select the certificate in the [App Service Certificates](#) page, then select **Rekey** and **Sync** from the left navigation.

Click **Rekey** to start the process. This process can take 1-10 minutes to complete.

The screenshot shows the Azure portal interface for managing App Service certificates. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Settings, Certificate Configuration, Auto Renew Settings, Timeline, Rekey and Sync (which is highlighted with a red box), Export Certificate, Properties, Locks, Automation script, Support + troubleshooting, FAQs, and New support request. At the top, there are buttons for Refresh, Rekey (highlighted with a red box), and Sync. The main area has a heading 'Rekey Certificate' with a key icon. It explains that rekeying will roll the certificate with a new one from the certificate authority. Below that is a note: 'Certificate rekey operations are free and rekey does not incur additional charges.' There's also a 'Linked Private Certificate' section with a 'Linked' button and some descriptive text about linked private certificates. At the bottom, there's a table for 'Current Certificate Thumbprint' with columns for STATUS, LINKED PRIVATE CERTIFI..., RESOURCE GROUP, and THUMBPRINT. A single row shows 'Healthy' status, 'myResourceGro...', and a blurred thumbprint value.

Rekeying your certificate rolls the certificate with a new certificate issued from the certificate authority.

Once the rekey operation is complete, click **Sync**. The sync operation automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

#### NOTE

If you don't click **Sync**, App Service automatically syncs your certificate within 24 hours.

### Export certificate

Because an App Service Certificate is a [Key Vault secret](#), you can export a PFX copy of it and use it for other Azure services or outside of Azure.

To export the App Service Certificate as a PFX file, run the following commands in the [Cloud Shell](#). You can also run it locally if you [installed Azure CLI](#). Replace the placeholders with the names you used when you [created the App Service certificate](#).

```

secretname=$(az resource show \
--resource-group <group-name> \
--resource-type "Microsoft.CertificateRegistration/certificateOrders" \
--name <app-service-cert-name> \
--query "properties.certificates.<app-service-cert-name>.keyVaultSecretName" \
--output tsv)

az keyvault secret download \
--file appservicecertificate.pfx \
--vault-name <key-vault-name> \
--name $secretname \
--encoding base64

```

The downloaded *appservicecertificate.pfx* file is a raw PKCS12 file that contains both the public and private certificates. In each prompt, use an empty string for the import password and the PEM pass phrase.

### Delete certificate

Deletion of an App Service certificate is final and irreversible. Deletion of a App Service Certificate resource results in the certificate being revoked. Any binding in App Service with this certificate becomes invalid. To prevent accidental deletion, Azure puts a lock on the certificate. To delete an App Service certificate, you must first remove the delete lock on the certificate.

Select the certificate in the [App Service Certificates](#) page, then select **Locks** in the left navigation.

Find the lock on your certificate with the lock type **Delete**. To the right of it, select **Delete**.

Lock name	Lock type	Scope	Notes	Actions
contoso	Delete	contoso	Deleting this App Service certi...	<a href="#">Edit</a> <a href="#">Delete</a>

Now you can delete the App Service certificate. From the left navigation, select **Overview** > **Delete**. In the confirmation dialog, type the certificate name and select **OK**.

## Automate with scripts

### Azure CLI

```

#!/bin/bash

fqdn=<replace-with-www.{yourdomain}>
pfxPath=<replace-with-path-to-your-.PFX-file>
pfxPassword=<replace-with-your=.PFX-password>
resourceGroup=myResourceGroup
webappName=mywebapp$RANDOM

Create a resource group.
az group create --location westeurope --name $resourceGroup

Create an App Service plan in Basic tier (minimum required by custom domains).
az appservice plan create --name $webappName --resource-group $resourceGroup --sku B1

Create a web app.
az webapp create --name $webappName --resource-group $resourceGroup \
--plan $webappName

echo "Configure a CNAME record that maps $fqdn to $webappName.azurewebsites.net"
read -p "Press [Enter] key when ready ..."

Before continuing, go to your DNS configuration UI for your custom domain and follow the
instructions at https://aka.ms/appservicecustomdns to configure a CNAME record for the
hostname "www" and point it to your web app's default domain name.

Map your prepared custom domain name to the web app.
az webapp config hostname add --webapp-name $webappName --resource-group $resourceGroup \
--hostname $fqdn

Upload the SSL certificate and get the thumbprint.
thumbprint=$(az webapp config ssl upload --certificate-file $pfxPath \
--certificate-password $pfxPassword --name $webappName --resource-group $resourceGroup \
--query thumbprint --output tsv)

Binds the uploaded SSL certificate to the web app.
az webapp config ssl bind --certificate-thumbprint $thumbprint --ssl-type SNI \
--name $webappName --resource-group $resourceGroup

echo "You can now browse to https://$fqdn"

```

## PowerShell

```

$fqdn=<Replace with your custom domain name>
$pfxPath=<Replace with path to your .PFX file>
$pfxPassword=<Replace with your .PFX password>
$webappname="mywebapp$(Get-Random)"
$location="West Europe"

Create a resource group.
New-AzResourceGroup -Name $webappname -Location $location

Create an App Service plan in Free tier.
New-AzAppServicePlan -Name $webappname -Location $location `
-ResourceGroupName $webappname -Tier Free

Create a web app.
New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname `
-ResourceGroupName $webappname

Write-Host "Configure a CNAME record that maps $fqdn to $webappname.azurewebsites.net"
Read-Host "Press [Enter] key when ready ..."

Before continuing, go to your DNS configuration UI for your custom domain and follow the
instructions at https://aka.ms/appservicecustomdns to configure a CNAME record for the
hostname "www" and point it to your web app's default domain name.

Upgrade App Service plan to Basic tier (minimum required by custom SSL certificates)
Set-AzAppServicePlan -Name $webappname -ResourceGroupName $webappname `
-Tier Basic

Add a custom domain name to the web app.
Set-AzWebApp -Name $webappname -ResourceGroupName $webappname `
-HostNames @($fqdn,"$webappname.azurewebsites.net")

Upload and bind the SSL certificate to the web app.
New-AzWebAppSSLBinding -WebAppName $webappname -ResourceGroupName $webappname -Name $fqdn `
-CertificateFilePath $pfxPath -CertificatePassword $pfxPassword -SslState SniEnabled

```

## More resources

- [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [Use a TLS/SSL certificate in your code in Azure App Service](#)
- [FAQ : App Service Certificates](#)

# Set up Azure App Service access restrictions

11/2/2021 • 9 minutes to read • [Edit Online](#)

By setting up access restrictions, you can define a priority-ordered allow/deny list that controls network access to your app. The list can include IP addresses or Azure Virtual Network subnets. When there are one or more entries, an implicit *deny all* exists at the end of the list.

The access restriction capability works with all Azure App Service-hosted workloads. The workloads can include web apps, API apps, Linux apps, Linux container apps, and Functions.

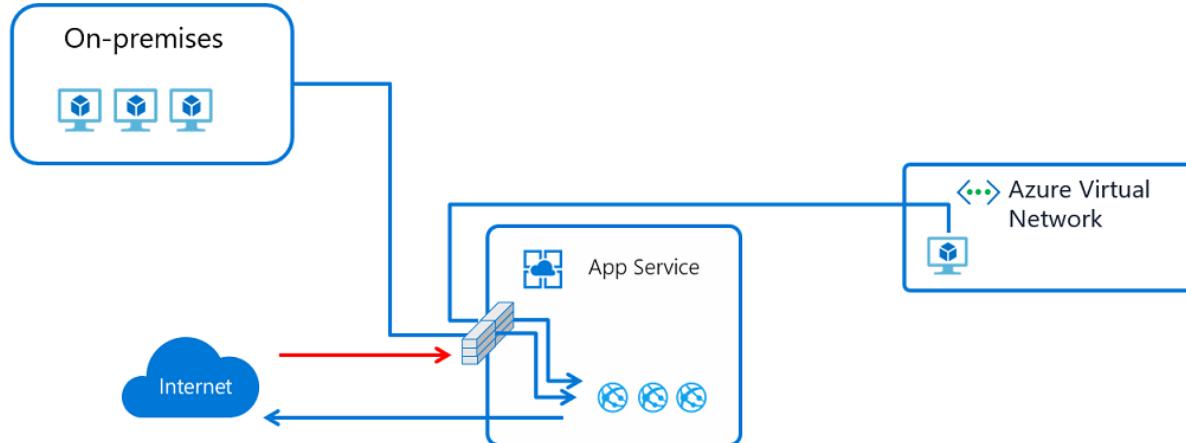
When a request is made to your app, the FROM address is evaluated against the rules in your access restriction list. If the FROM address is in a subnet that's configured with service endpoints to Microsoft.Web, the source subnet is compared against the virtual network rules in your access restriction list. If the address isn't allowed access based on the rules in the list, the service replies with an [HTTP 403](#) status code.

The access restriction capability is implemented in the App Service front-end roles, which are upstream of the worker hosts where your code runs. Therefore, access restrictions are effectively network access-control lists (ACLs).

The ability to restrict access to your web app from an Azure virtual network is enabled by [service endpoints](#). With service endpoints, you can restrict access to a multi-tenant service from selected subnets. It doesn't work to restrict traffic to apps that are hosted in an App Service Environment. If you're in an App Service Environment, you can control access to your app by applying IP address rules.

## NOTE

The service endpoints must be enabled both on the networking side and for the Azure service that they're being enabled with. For a list of Azure services that support service endpoints, see [Virtual Network service endpoints](#).



## Manage access restriction rules in the portal

To add an access restriction rule to your app, do the following:

1. Sign in to the Azure portal.
2. On the left pane, select **Networking**.
3. On the **Networking** pane, under **Access Restrictions**, select **Configure Access Restrictions**.

#### 4. On the **Access Restrictions** page, review the list of access restriction rules that are defined for your app.

PRIORITY	NAME	SOURCE	ENDPOINT STATUS	ACTION
100	IP example rule	122.133.144.0/24		<span>Allow</span>
150	deny example	122.133.144.32/28		<span>Deny</span>
200	test rule	networking-demos-vnet/simple-se-su...	Enabled	<span>Allow</span>
2147483647	Deny all	Any		<span>Deny</span>

The list displays all the current restrictions that are applied to the app. If you have a virtual network restriction on your app, the table shows whether the service endpoints are enabled for Microsoft.Web. If no restrictions are defined on your app, the app is accessible from anywhere.

#### Add an access restriction rule

To add an access restriction rule to your app, on the **Access Restrictions** pane, select **Add rule**. After you add a rule, it becomes effective immediately.

Rules are enforced in priority order, starting from the lowest number in the **Priority** column. An implicit *deny all* is in effect after you add even a single rule.

On the **Add Access Restriction** pane, when you create a rule, do the following:

- Under **Action**, select either **Allow** or **Deny**.

- Optionally, enter a name and description of the rule.
- In the **Priority** box, enter a priority value.
- In the **Type** drop-down list, select the type of rule.

The different types of rules are described in the following sections.

#### NOTE

- There is a limit of 512 access restriction rules. If you require more than 512 access restriction rules, we suggest that you consider installing a standalone security product, such as Azure Front Door, Azure App Gateway, or an alternative WAF.

#### Set an IP address-based rule

Follow the procedure as outlined in the preceding section, but with the following addition:

- For step 4, in the **Type** drop-down list, select **IPv4** or **IPv6**.

Specify the **IP Address Block** in Classless Inter-Domain Routing (CIDR) notation for both the IPv4 and IPv6 addresses. To specify an address, you can use something like `1.2.3.4/32`, where the first four octets represent your IP address and `/32` is the mask. The IPv4 CIDR notation for all addresses is `0.0.0.0/0`. To learn more about CIDR notation, see [Classless Inter-Domain Routing](#).

#### Set a service endpoint-based rule

- For step 4, in the **Type** drop-down list, select **Virtual Network**.

Specify the **Subscription**, **Virtual Network**, and **Subnet** drop-down lists, matching what you want to restrict access to.

By using service endpoints, you can restrict access to selected Azure virtual network subnets. If service endpoints aren't already enabled with Microsoft.Web for the subnet that you selected, they'll be automatically enabled unless you select the **Ignore missing Microsoft.Web service endpoints** check box. The scenario where you might want to enable service endpoints on the app but not the subnet depends mainly on whether you have the permissions to enable them on the subnet.

If you need someone else to enable service endpoints on the subnet, select the **Ignore missing Microsoft.Web service endpoints** check box. Your app will be configured for service endpoints in anticipation of having them enabled later on the subnet.

You can't use service endpoints to restrict access to apps that run in an App Service Environment. When your app is in an App Service Environment, you can control access to it by applying IP access rules.

With service endpoints, you can configure your app with application gateways or other web application firewall (WAF) devices. You can also configure multi-tier applications with secure back ends. For more information, see [Networking features and App Service](#) and [Application Gateway integration with service endpoints](#).

#### NOTE

- Service endpoints aren't currently supported for web apps that use IP-based TLS/SSL bindings with a virtual IP (VIP).

#### Set a service tag-based rule

- For step 4, in the **Type** drop-down list, select **Service Tag**.

Each service tag represents a list of IP ranges from Azure services. A list of these services and links to the specific ranges can be found in the [service tag documentation](#).

All available service tags are supported in access restriction rules. For simplicity, only a list of the most common tags are available through the Azure portal. Use Azure Resource Manager templates or scripting to configure

more advanced rules like regional scoped rules. These are the tags available through Azure portal:

- ActionGroup
- ApplicationInsightsAvailability
- AzureCloud
- AzureCognitiveSearch
- AzureEventGrid
- AzureFrontDoor.Backend
- AzureMachineLearning
- AzureTrafficManager
- LogicApps

## Edit a rule

1. To begin editing an existing access restriction rule, on the **Access Restrictions** page, select the rule you want to edit.
2. On the **Edit Access Restriction** pane, make your changes, and then select **Update rule**. Edits are effective immediately, including changes in priority ordering.



### NOTE

When you edit a rule, you can't switch between rule types.

## Delete a rule

To delete a rule, on the **Access Restrictions** page, select the ellipsis (...) next to the rule you want to delete, and then select **Remove**.

The screenshot shows the 'Access Restrictions' blade for an app service named 'vnet-integration-app'. At the top, there's a search bar with the placeholder 'vnet-integration-app.azurewebsites.net' and a 'vnet-integration-app.scm.azurewebsites.net' dropdown. Below the search bar is a button labeled '+ Add rule'. The main table lists four rules:

PRIORITY	NAME	SOURCE	ENDPOINT STATUS	ACTION	... (More Options)
100	IP example rule	122.133.144.0/24		<span>Allow</span>	...
150	deny example	122.133.144.32/28		<span>Deny</span>	<span>Remove</span>
200	test rule	networking-demos-vnet/simple-se-sub...	Enabled	<span>Allow</span>	...
2147483647	Deny all	Any		<span>Deny</span>	...

## Access restriction advanced scenarios

The following sections describe some advanced scenarios using access restrictions.

### Filter by http header

As part of any rule, you can add additional http header filters. The following http header names are supported:

- X-Forwarded-For
- X-Forwarded-Host

- X-Azure-FDID
- X-FD-HealthProbe

For each header name, you can add up to eight values separated by comma. The http header filters are evaluated after the rule itself and both conditions must be true for the rule to apply.

### Multi-source rules

Multi-source rules allow you to combine up to eight IP ranges or eight Service Tags in a single rule. You might use this if you have more than 512 IP ranges or you want to create logical rules where multiple IP ranges are combined with a single http header filter.

Multi-source rules are defined the same way you define single-source rules, but with each range separated with comma.

PowerShell example:

```
Add-AzWebAppAccessRestrictionRule -ResourceGroupName "ResourceGroup" -WebAppName "AppName" `
-Name "Multi-source rule" -IpAddress "192.168.1.0/24,192.168.10.0/24,192.168.100.0/24" `
-Priority 100 -Action Allow
```

### Block a single IP address

When you add your first access restriction rule, the service adds an explicit *Deny all* rule with a priority of 2147483647. In practice, the explicit *Deny all* rule is the final rule to be executed, and it blocks access to any IP address that's not explicitly allowed by an *Allow* rule.

For a scenario where you want to explicitly block a single IP address or a block of IP addresses, but allow access to everything else, add an explicit *Allow All* rule.

PRIORITY	NAME	SOURCE	ENDPOINT STATUS	ACTION
200	Block Me	131.107.147/32		<span style="color:red;">X</span> Deny
300	Allow All	0.0.0.0/0		<span style="color:green;">✓</span> Allow
2147483647	Deny all	Any		<span style="color:red;">X</span> Deny

### Restrict access to an SCM site

In addition to being able to control access to your app, you can restrict access to the SCM site that's used by your app. The SCM site is both the web deploy endpoint and the Kudu console. You can assign access restrictions to the SCM site from the app separately or use the same set of restrictions for both the app and the SCM site. When you select the **Same restrictions as <app name>** check box, everything is blanked out. If you clear the check box, your SCM site settings are reapplied.

The screenshot shows the 'Access Restrictions' page in the Azure portal. At the top, there are buttons for 'Remove' and 'Refresh'. Below that, a section titled 'Access Restrictions' explains what they are used for. A table lists the current rules:

PRIORITY	NAME	SOURCE	ENDPOINT STATUS	ACTION
1	Allow all	Any		<span style="color: green;">Allow</span>

## Restrict access to a specific Azure Front Door instance

Traffic from Azure Front Door to your application originates from a well known set of IP ranges defined in the `AzureFrontDoorBackend` service tag. Using a service tag restriction rule, you can restrict traffic to only originate from Azure Front Door. To ensure traffic only originates from your specific instance, you will need to further filter the incoming requests based on the unique http header that Azure Front Door sends.

PowerShell example:

```
$afd = Get-AzFrontDoor -Name "MyFrontDoorInstanceName"
Add-AzWebAppAccessRestrictionRule -ResourceGroupName "ResourceGroup" -WebAppName "AppName" `
 -Name "Front Door example rule" -Priority 100 -Action Allow -ServiceTag AzureFrontDoor.Backend `
 -HttpHeader @{
 'x-azure-fdid' = $afd.FrontDoorId}
```

## Manage access restriction rules programmatically

You can add access restrictions programmatically by doing either of the following:

- Use [the Azure CLI](#). For example:

```
az webapp config access-restriction add --resource-group ResourceGroup --name AppName \
 --rule-name 'IP example rule' --action Allow --ip-address 122.133.144.0/24 --priority 100
```

### NOTE

Working with service tags, http headers or multi-source rules in Azure CLI requires at least version 2.23.0. You can verify the version of the installed module with: `az version`

- Use [Azure PowerShell](#). For example:

```
Add-AzWebAppAccessRestrictionRule -ResourceGroupName "ResourceGroup" -WebAppName "AppName" `
 -Name "Ip example rule" -Priority 100 -Action Allow -IpAddress 122.133.144.0/24
```

## NOTE

Working with service tags, http headers or multi-source rules in Azure PowerShell requires at least version 5.7.0.

You can verify the version of the installed module with: `Get-InstalledModule -Name Az`

You can also set values manually by doing either of the following:

- Use an [Azure REST API](#) PUT operation on the app configuration in Azure Resource Manager. The location for this information in Azure Resource Manager is:

```
management.azure.com/subscriptions/subscription ID/resourceGroups/resource groups/providers/Microsoft.Web/sites/web app name/config/web?api-version=2020-06-01
```

- Use a Resource Manager template. As an example, you can use resources.azure.com and edit the ipSecurityRestrictions block to add the required JSON.

The JSON syntax for the earlier example is:

```
{
 "properties": {
 "ipSecurityRestrictions": [
 {
 "ipAddress": "122.133.144.0/24",
 "action": "Allow",
 "priority": 100,
 "name": "IP example rule"
 }
]
 }
}
```

The JSON syntax for an advanced example using service tag and http header restriction is:

```
{
 "properties": {
 "ipSecurityRestrictions": [
 {
 "ipAddress": "AzureFrontDoor.Backend",
 "tag": "ServiceTag",
 "action": "Allow",
 "priority": 100,
 "name": "Azure Front Door example",
 "headers": {
 "x-azure-fdid": [
 "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
]
 }
 }
]
 }
}
```

## Set up Azure Functions access restrictions

Access restrictions are also available for function apps with the same functionality as App Service plans. When you enable access restrictions, you also disable the Azure portal code editor for any disallowed IPs.

## Next steps

[Access restrictions for Azure Functions](#)

[Application Gateway integration with service endpoints](#)

# How to use managed identities for App Service and Azure Functions

11/2/2021 • 16 minutes to read • [Edit Online](#)

This topic shows you how to create a managed identity for App Service and Azure Functions applications and how to use it to access other resources.

## IMPORTANT

Managed identities for App Service and Azure Functions won't behave as expected if your app is migrated across subscriptions/tenants. The app needs to obtain a new identity, which is done by disabling and re-enabling the feature. See [Removing an identity](#) below. Downstream resources also need to have access policies updated to use the new identity.

## NOTE

Managed identities are not available for [apps deployed in Azure Arc](#).

A managed identity from Azure Active Directory (Azure AD) allows your app to easily access other Azure AD-protected resources such as Azure Key Vault. The identity is managed by the Azure platform and does not require you to provision or rotate any secrets. For more about managed identities in Azure AD, see [Managed identities for Azure resources](#).

Your application can be granted two types of identities:

- A **system-assigned identity** is tied to your application and is deleted if your app is deleted. An app can only have one system-assigned identity.
- A **user-assigned identity** is a standalone Azure resource that can be assigned to your app. An app can have multiple user-assigned identities.

## Add a system-assigned identity

Creating an app with a system-assigned identity requires an additional property to be set on the application.

### Using the Azure portal

To set up a managed identity in the portal, you will first create an application as normal and then enable the feature.

1. Create an app in the portal as you normally would. Navigate to it in the portal.
2. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.
3. Select **Identity**.
4. Within the **System assigned** tab, switch **Status** to **On**. Click **Save**.

The screenshot shows the Azure portal interface for managing an App Service. The main title bar says 'systemassigned-linux' and the URL is 'https://portal.azure.com/#@contoso.onmicrosoft.com/resource/subscriptions/01234567-89ab-cdef-0123-456789abcdef/resourceGroups...'. The top navigation bar includes 'Microsoft Azure', a search bar, and a user profile for 'meganb@contoso.com CONTOSO'. Below the search bar, the breadcrumb navigation shows 'Home > App Services > systemassigned-linux - Identity'. The main content area is titled 'systemassigned-linux-Identity' and 'App Service'. On the left, a sidebar lists several settings: 'Authentication / Authorization...', 'Application Insights', 'Identity' (which is highlighted with a red box), 'Backups', 'Custom domains', 'TLS/SSL settings', 'Networking', and 'Scale up (App Service plan)'. The main pane displays information about system assigned managed identities, with a note that each resource can only have one system assigned managed identity. It includes a 'Status' section with a toggle switch between 'Off' and 'On', where 'On' is highlighted with a red box. At the bottom of the main pane are buttons for 'Save', 'Discard', 'Refresh', and 'Got feedback?'. A vertical scrollbar is on the right side of the main content area.

#### NOTE

To find the managed identity for your web app or slot app in the Azure portal, under **Enterprise applications**, look in the **User settings** section. Usually, the slot name is similar to <app name>/slots/<slot name>.

## Using the Azure CLI

To set up a managed identity using the Azure CLI, you will need to use the `az webapp identity assign` command against an existing application. You have three options for running the examples in this section:

- Use [Azure Cloud Shell](#) from the Azure portal.
- Use the embedded Azure Cloud Shell via the "Try It" button, located in the top-right corner of each code block below.
- [Install the latest version of Azure CLI](#) (2.0.31 or later) if you prefer to use a local CLI console.

The following steps will walk you through creating a web app and assigning it an identity using the CLI:

1. If you're using the Azure CLI in a local console, first sign in to Azure using `az login`. Use an account that's associated with the Azure subscription under which you would like to deploy the application:

```
az login
```

2. Create a web application using the CLI. For more examples of how to use the CLI with App Service, see [App Service CLI samples](#):

```
az group create --name myResourceGroup --location westus
az appservice plan create --name myPlan --resource-group myResourceGroup --sku S1
az webapp create --name myApp --resource-group myResourceGroup --plan myPlan
```

3. Run the `identity assign` command to create the identity for this application:

```
az webapp identity assign --name myApp --resource-group myResourceGroup
```

## Using Azure PowerShell

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

The following steps will walk you through creating an app and assigning it an identity using Azure PowerShell. The instructions for creating a web app and a function app are different.

### Using Azure PowerShell for a web app

1. If needed, install the Azure PowerShell using the instructions found in the [Azure PowerShell guide](#), and then run `Login-AzAccount` to create a connection with Azure.
2. Create a web application using Azure PowerShell. For more examples of how to use Azure PowerShell with App Service, see [App Service PowerShell samples](#):

```
Create a resource group.
New-AzResourceGroup -Name $resourceGroupName -Location $location

Create an App Service plan in Free tier.
New-AzAppServicePlan -Name $webappname -Location $location -ResourceGroupName $resourceGroupName -
Tier Free

Create a web app.
New-AzWebApp -Name $webappname -Location $location -AppServicePlan $webappname -ResourceGroupName
$resourceGroupName
```

3. Run the `Set-AzWebApp -AssignIdentity` command to create the identity for this application:

```
Set-AzWebApp -AssignIdentity $true -Name $webappname -ResourceGroupName $resourceGroupName
```

### Using Azure PowerShell for a function app

1. If needed, install the Azure PowerShell using the instructions found in the [Azure PowerShell guide](#), and then run `Login-AzAccount` to create a connection with Azure.
2. Create a function app using Azure PowerShell. For more examples of how to use Azure PowerShell with Azure Functions, see the [Az.Functions reference](#):

```
Create a resource group.
New-AzResourceGroup -Name $resourceGroupName -Location $location

Create a storage account.
New-AzStorageAccount -Name $storageAccountName -ResourceGroupName $resourceGroupName -SkuName $sku

Create a function app with a system-assigned identity.
New-AzFunctionApp -Name $functionAppName -ResourceGroupName $resourceGroupName -Location $location -
StorageAccountName $storageAccountName -Runtime $runtime -IdentityType SystemAssigned
```

You can also update an existing function app using `Update-AzFunctionApp` instead.

### Using an Azure Resource Manager template

An Azure Resource Manager template can be used to automate deployment of your Azure resources. To learn more about deploying to App Service and Functions, see [Automating resource deployment in App Service](#) and [Automating resource deployment in Azure Functions](#).

Any resource of type `Microsoft.Web/sites` can be created with an identity by including the following property in

the resource definition:

```
"identity": {
 "type": "SystemAssigned"
}
```

#### NOTE

An application can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned,UserAssigned`

Adding the system-assigned type tells Azure to create and manage the identity for your application.

For example, a web app might look like the following:

```
{
 "apiVersion": "2016-08-01",
 "type": "Microsoft.Web/sites",
 "name": "[variables('appName')]",
 "location": "[resourceGroup().location]",
 "identity": {
 "type": "SystemAssigned"
 },
 "properties": {
 "name": "[variables('appName')]",
 "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]",
 "hostingEnvironment": "",
 "clientAffinityEnabled": false,
 "alwaysOn": true
 },
 "dependsOn": [
 "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]"
]
}
```

When the site is created, it has the following additional properties:

```
"identity": {
 "type": "SystemAssigned",
 "tenantId": "<TENANTID>",
 "principalId": "<PRINCIPALID>"
}
```

The `tenantId` property identifies what Azure AD tenant the identity belongs to. The `principalId` is a unique identifier for the application's new identity. Within Azure AD, the service principal has the same name that you gave to your App Service or Azure Functions instance.

If you need to reference these properties in a later stage in the template, you can do so via the [reference\(\)](#) [template function](#) with the `'Full'` flag, as in this example:

```
{
 "tenantId": "[reference(resourceId('Microsoft.Web/sites', variables('appName')), '2018-02-01',
 'Full').identity.tenantId]",
 "objectId": "[reference(resourceId('Microsoft.Web/sites', variables('appName')), '2018-02-01',
 'Full').identity.principalId]",
}
```

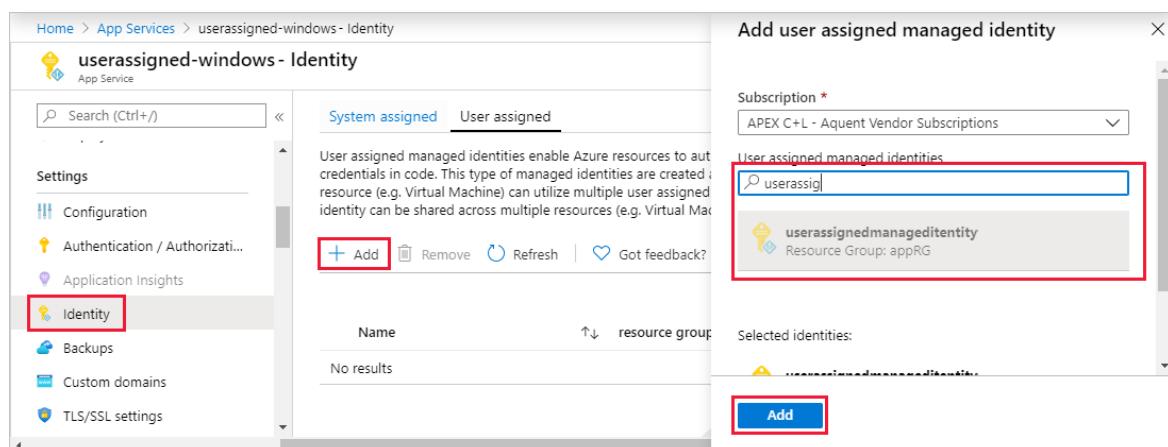
# Add a user-assigned identity

Creating an app with a user-assigned identity requires that you create the identity and then add its resource identifier to your app config.

## Using the Azure portal

First, you'll need to create a user-assigned identity resource.

1. Create a user-assigned managed identity resource according to [these instructions](#).
2. Create an app in the portal as you normally would. Navigate to it in the portal.
3. If using a function app, navigate to **Platform features**. For other app types, scroll down to the **Settings** group in the left navigation.
4. Select **Identity**.
5. Within the **User assigned** tab, click **Add**.
6. Search for the identity you created earlier and select it. Click **Add**.



The screenshot shows the Azure portal interface for managing identities. On the left, there's a sidebar with options like 'Settings', 'Configuration', 'Authentication / Authorization', 'Application Insights', and 'Identity' (which is highlighted with a red box). The main area has tabs for 'System assigned' and 'User assigned' (which is selected). Below the tabs, there's a brief description of user-assigned identities. At the top right of the main area, there's a 'Subscription' dropdown set to 'APEX C+L - Aquent Vendor Subscriptions'. Below that is a search bar with 'userassig' typed into it. A list of identities is displayed, with one entry 'userassignedmanagedidentity' from 'Resource Group: appRG' highlighted with a red box. At the bottom right of the main area, there's a large blue 'Add' button.

## Using Azure PowerShell

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

The following steps will walk you through creating an app and assigning it an identity using Azure PowerShell.

### NOTE

The current version of the Azure PowerShell commandlets for Azure App Service do not support user-assigned identities. The below instructions are for Azure Functions.

1. If needed, install the Azure PowerShell using the instructions found in the [Azure PowerShell guide](#), and then run `Login-AzAccount` to create a connection with Azure.
2. Create a function app using Azure PowerShell. For more examples of how to use Azure PowerShell with Azure Functions, see the [Az.Functions reference](#). The below script also makes use of `New-AzUserAssignedIdentity` which must be installed separately as per [Create, list or delete a user-assigned managed identity using Azure PowerShell](#).

```

Create a resource group.
New-AzResourceGroup -Name $resourceGroupName -Location $location

Create a storage account.
New-AzStorageAccount -Name $storageAccountName -ResourceGroupName $resourceGroupName -SkuName $sku

Create a user-assigned identity. This requires installation of the "Az.ManagedServiceIdentity" module.
$userAssignedIdentity = New-AzUserAssignedIdentity -Name $userAssignedIdentityName -ResourceGroupName $resourceGroupName

Create a function app with a user-assigned identity.
New-AzFunctionApp -Name $functionAppName -ResourceGroupName $resourceGroupName -Location $location - StorageAccountName $storageAccountName -Runtime $runtime -IdentityType UserAssigned -IdentityId $userAssignedIdentity.Id

```

You can also update an existing function app using `Update-AzFunctionApp` instead.

## Using an Azure Resource Manager template

An Azure Resource Manager template can be used to automate deployment of your Azure resources. To learn more about deploying to App Service and Functions, see [Automating resource deployment in App Service](#) and [Automating resource deployment in Azure Functions](#).

Any resource of type `Microsoft.Web/sites` can be created with an identity by including the following block in the resource definition, replacing `<RESOURCEID>` with the resource ID of the desired identity:

```

"identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "<RESOURCEID>": {}
 }
}

```

### NOTE

An application can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned,UserAssigned`

Adding the user-assigned type tells Azure to use the user-assigned identity specified for your application.

For example, a web app might look like the following:

```
{
 "apiVersion": "2016-08-01",
 "type": "Microsoft.Web/sites",
 "name": "[variables('appName')]",
 "location": "[resourceGroup().location]",
 "identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "[resourceId('Microsoft.ManagedIdentity/userAssignedIdentities', variables('identityName'))]":
 }
 },
 "properties": {
 "name": "[variables('appName')]",
 "serverFarmId": "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]",
 "hostingEnvironment": "",
 "clientAffinityEnabled": false,
 "alwaysOn": true
 },
 "dependsOn": [
 "[resourceId('Microsoft.Web/serverfarms', variables('hostingPlanName'))]",
 "[resourceId('Microsoft.ManagedIdentity/userAssignedIdentities', variables('identityName'))]"
]
}
}
```

When the site is created, it has the following additional properties:

```
"identity": {
 "type": "UserAssigned",
 "userAssignedIdentities": {
 "<RESOURCEID>": {
 "principalId": "<PRINCIPALID>",
 "clientId": "<CLIENTID>"
 }
 }
}
```

The principalId is a unique identifier for the identity that's used for Azure AD administration. The clientId is a unique identifier for the application's new identity that's used for specifying which identity to use during runtime calls.

## Obtain tokens for Azure resources

An app can use its managed identity to get tokens to access other resources protected by Azure AD, such as Azure Key Vault. These tokens represent the application accessing the resource, and not any specific user of the application.

You may need to configure the target resource to allow access from your application. For example, if you request a token to access Key Vault, you need to make sure you have added an access policy that includes your application's identity. Otherwise, your calls to Key Vault will be rejected, even if they include the token. To learn more about which resources support Azure Active Directory tokens, see [Azure services that support Azure AD authentication](#).

### IMPORTANT

The back-end services for managed identities maintain a cache per resource URI for around 24 hours. If you update the access policy of a particular target resource and immediately retrieve a token for that resource, you may continue to get a cached token with outdated permissions until that token expires. There's currently no way to force a token refresh.

There is a simple REST protocol for obtaining a token in App Service and Azure Functions. This can be used for all applications and languages. For .NET and Java, the Azure SDK provides an abstraction over this protocol and facilitates a local development experience.

## Using the REST protocol

### NOTE

An older version of this protocol, using the "2017-09-01" API version, used the `secret` header instead of `X-IDENTITY-HEADER` and only accepted the `clientid` property for user-assigned. It also returned the `expires_on` in a timestamp format. `MSI_ENDPOINT` can be used as an alias for `IDENTITY_ENDPOINT`, and `MSI_SECRET` can be used as an alias for `IDENTITY_HEADER`. This version of the protocol is currently required for Linux Consumption hosting plans.

An app with a managed identity has two environment variables defined:

- `IDENTITY_ENDPOINT` - the URL to the local token service.
- `IDENTITY_HEADER` - a header used to help mitigate server-side request forgery (SSRF) attacks. The value is rotated by the platform.

The `IDENTITY_ENDPOINT` is a local URL from which your app can request tokens. To get a token for a resource, make an HTTP GET request to this endpoint, including the following parameters:

PARAMETER NAME	IN	DESCRIPTION
resource	Query	The Azure AD resource URI of the resource for which a token should be obtained. This could be one of the <a href="#">Azure services that support Azure AD authentication</a> or any other resource URI.
api-version	Query	The version of the token API to be used. Please use "2019-08-01" or later (unless using Linux Consumption, which currently only offers "2017-09-01" - see note above).
X-IDENTITY-HEADER	Header	The value of the <code>IDENTITY_HEADER</code> environment variable. This header is used to help mitigate server-side request forgery (SSRF) attacks.
client_id	Query	(Optional) The client ID of the user-assigned identity to be used. Cannot be used on a request that includes <code>principal_id</code> , <code>mi_res_id</code> , or <code>object_id</code> . If all ID parameters ( <code>client_id</code> , <code>principal_id</code> , <code>object_id</code> , and <code>mi_res_id</code> ) are omitted, the system-assigned identity is used.

PARAMETER NAME	IN	DESCRIPTION
principal_id	Query	(Optional) The principal ID of the user-assigned identity to be used. <code>object_id</code> is an alias that may be used instead. Cannot be used on a request that includes <code>client_id</code> , <code>mi_res_id</code> , or <code>object_id</code> . If all ID parameters ( <code>client_id</code> , <code>principal_id</code> , <code>object_id</code> , and <code>mi_res_id</code> ) are omitted, the system-assigned identity is used.
mi_res_id	Query	(Optional) The Azure resource ID of the user-assigned identity to be used. Cannot be used on a request that includes <code>principal_id</code> , <code>client_id</code> , or <code>object_id</code> . If all ID parameters ( <code>client_id</code> , <code>principal_id</code> , <code>object_id</code> , and <code>mi_res_id</code> ) are omitted, the system-assigned identity is used.

#### IMPORTANT

If you are attempting to obtain tokens for user-assigned identities, you must include one of the optional properties. Otherwise the token service will attempt to obtain a token for a system-assigned identity, which may or may not exist.

A successful 200 OK response includes a JSON body with the following properties:

PROPERTY NAME	DESCRIPTION
access_token	The requested access token. The calling web service can use this token to authenticate to the receiving web service.
client_id	The client ID of the identity that was used.
expires_on	The timespan when the access token expires. The date is represented as the number of seconds from "1970-01-01T0:0:0Z UTC" (corresponds to the token's <code>exp</code> claim).
not_before	The timespan when the access token takes effect, and can be accepted. The date is represented as the number of seconds from "1970-01-01T0:0:0Z UTC" (corresponds to the token's <code>nbf</code> claim).
resource	The resource the access token was requested for, which matches the <code>resource</code> query string parameter of the request.
token_type	Indicates the token type value. The only type that Azure AD supports is Bearer. For more information about bearer tokens, see <a href="#">The OAuth 2.0 Authorization Framework: Bearer Token Usage (RFC 6750)</a> .

This response is the same as the [response for the Azure AD service-to-service access token request](#).

## REST protocol examples

An example request might look like the following:

```
GET /MSI/token?resource=https://vault.azure.net&api-version=2019-08-01 HTTP/1.1
Host: localhost:4141
X-IDENTITY-HEADER: 853b9a84-5bfa-4b22-a3f3-0b9a43d9ad8a
```

And a sample response might look like the following:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
 "access_token": "eyJ0eXAi...",
 "expires_on": "1586984735",
 "resource": "https://vault.azure.net",
 "token_type": "Bearer",
 "client_id": "5E29463D-71DA-4FE0-8E69-999B57DB23B0"
}
```

## Code examples

- [.NET](#)
- [JavaScript](#)
- [Python](#)
- [PowerShell](#)

### TIP

For .NET languages, you can also use [Microsoft.Azure.Services.AppAuthentication](#) instead of crafting this request yourself.

```
private readonly HttpClient _client;
// ...
public async Task<HttpResponseMessage> GetToken(string resource) {
 var request = new HttpRequestMessage(HttpMethod.Get,
 String.Format("{0}/?resource={1}&api-version=2019-08-01",
 Environment.GetEnvironmentVariable("IDENTITY_ENDPOINT"), resource));
 request.Headers.Add("X-IDENTITY-HEADER", Environment.GetEnvironmentVariable("IDENTITY_HEADER"));
 return await _client.SendAsync(request);
}
```

## Using the Microsoft.Azure.Services.AppAuthentication library for .NET

For .NET applications and functions, the simplest way to work with a managed identity is through the [Microsoft.Azure.Services.AppAuthentication](#) package. This library will also allow you to test your code locally on your development machine, using your user account from Visual Studio, the [Azure CLI](#), or Active Directory Integrated Authentication. When hosted in the cloud, it will default to using a system-assigned identity, but you can customize this behavior using a connection string environment variable which references the client ID of a user-assigned identity. For more on development options with this library, see the [Microsoft.Azure.Services.AppAuthentication reference](#). This section shows you how to get started with the library in your code.

1. Add references to the [Microsoft.Azure.Services.AppAuthentication](#) and any other necessary NuGet packages to your application. The below example also uses [Microsoft.Azure.KeyVault](#).

2. Add the following code to your application, modifying to target the correct resource. This example shows two ways to work with Azure Key Vault:

```
using Microsoft.Azure.Services.AppAuthentication;
using Microsoft.Azure.KeyVault;
// ...
var azureServiceTokenProvider = new AzureServiceTokenProvider();
string accessToken = await azureServiceTokenProvider.GetAccessTokenAsync("https://vault.azure.net");
// OR
var kv = new KeyVaultClient(new
KeyVaultClient.AuthenticationCallback(azureServiceTokenProvider.KeyVaultTokenCallback));
```

If you want to use a user-assigned managed identity, you can set the `AzureServicesAuthConnectionString` application setting to `RunAs=App;AppId=<clientId-guid>`. Replace `<clientId-guid>` with the client ID of the identity you want to use. You can define multiple such connection strings by using custom application settings and passing their values into the `AzureServiceTokenProvider` constructor.

```
var identityConnectionString1 = Environment.GetEnvironmentVariable("UA1_ConnectionString");
var azureServiceTokenProvider1 = new AzureServiceTokenProvider(identityConnectionString1);

var identityConnectionString2 = Environment.GetEnvironmentVariable("UA2_ConnectionString");
var azureServiceTokenProvider2 = new AzureServiceTokenProvider(identityConnectionString2);
```

To learn more about configuring `AzureServiceTokenProvider` and the operations it exposes, see the [Microsoft.Azure.Services.AppAuthentication reference](#) and the [App Service and KeyVault with MSI .NET sample](#).

## Using the Azure SDK for Java

For Java applications and functions, the simplest way to work with a managed identity is through the [Azure SDK for Java](#). This section shows you how to get started with the library in your code.

1. Add a reference to the [Azure SDK library](#). For Maven projects, you might add this snippet to the `dependencies` section of the project's POM file:

```
<dependency>
 <groupId>com.microsoft.azure</groupId>
 <artifactId>azure</artifactId>
 <version>1.23.0</version>
</dependency>
```

2. Use the `AppServiceMSICredentials` object for authentication. This example shows how this mechanism may be used for working with Azure Key Vault:

```
import com.microsoft.azure.AzureEnvironment;
import com.microsoft.azure.management.Azure;
import com.microsoft.azure.management.keyvault.Vault
//...
Azure azure = Azure.authenticate(new AppServiceMSICredentials(AzureEnvironment.AZURE))
 .withSubscription(subscriptionId);
Vault myKeyVault = azure.vaults().getByResourceGroup(resourceGroup, keyVaultName);
```

## Remove an identity

A system-assigned identity can be removed by disabling the feature using the portal, PowerShell, or CLI in the same way that it was created. User-assigned identities can be removed individually. To remove all identities, set the identity type to "None".

Removing a system-assigned identity in this way will also delete it from Azure AD. System-assigned identities are also automatically removed from Azure AD when the app resource is deleted.

To remove all identities in an [ARM template](#):

```
"identity": {
 "type": "None"
}
```

To remove all identities in Azure PowerShell (Azure Functions only):

```
Update an existing function app to have IdentityType "None".
Update-AzFunctionApp -Name $functionAppName -ResourceGroupName $resourceGroupName -IdentityType None
```

#### NOTE

There is also an application setting that can be set, WEBSITE\_DISABLE\_MSI, which just disables the local token service. However, it leaves the identity in place, and tooling will still show the managed identity as "on" or "enabled." As a result, use of this setting is not recommended.

## Next steps

- [Access SQL Database securely using a managed identity](#)
- [Access Azure Storage securely using a managed identity](#)
- [Call Microsoft Graph securely using a managed identity](#)
- [Connect securely to services with Key Vault secrets](#)

# Use Key Vault references for App Service and Azure Functions

11/2/2021 • 7 minutes to read • [Edit Online](#)

This topic shows you how to work with secrets from Azure Key Vault in your App Service or Azure Functions application without requiring any code changes. [Azure Key Vault](#) is a service that provides centralized secrets management, with full control over access policies and audit history.

## Granting your app access to Key Vault

In order to read secrets from Key Vault, you need to have a vault created and give your app permission to access it.

1. Create a key vault by following the [Key Vault quickstart](#).
2. Create a [managed identity](#) for your application.

Key Vault references will use the app's system assigned identity by default, but you can [specify a user-assigned identity](#).

3. Create an [access policy in Key Vault](#) for the application identity you created earlier. Enable the "Get" secret permission on this policy. Do not configure the "authorized application" or `applicationId` settings, as this is not compatible with a managed identity.

### Access network-restricted vaults

If your vault is configured with [network restrictions](#), you will also need to ensure that the application has network access.

1. Make sure the application has outbound networking capabilities configured, as described in [App Service networking features](#) and [Azure Functions networking options](#).

Linux applications attempting to use private endpoints additionally require that the app be explicitly configured to have all traffic route through the virtual network. This requirement will be removed in a forthcoming update. To set this, use the following CLI command:

```
az webapp config set --subscription <sub> -g <rg> -n <appname> --generic-configurations
'{"vnetRouteAllEnabled": true}'
```

2. Make sure that the vault's configuration accounts for the network or subnet through which your app will access it.

#### NOTE

Windows container currently does not support Key Vault references over VNet Integration.

### Access vaults with a user-assigned identity

Some apps need to reference secrets at creation time, when a system-assigned identity would not yet be available. In these cases, a user-assigned identity can be created and given access to the vault in advance.

Once you have granted permissions to the user-assigned identity, follow these steps:

1. [Assign the identity](#) to your application if you haven't already.
2. Configure the app to use this identity for Key Vault reference operations by setting the `keyVaultReferenceIdentity` property to the resource ID of the user-assigned identity.

```

userAssignedIdentityResourceId=$(az identity show -g MyResourceGroupName -n
MyUserAssignedIdentityName --query id -o tsv)
appResourceId=$(az webapp show -g MyResourceGroupName -n MyAppName --query id -o tsv)
az rest --method PATCH --uri "${appResourceId}?api-version=2021-01-01" --body "{'properties':
{'keyVaultReferenceIdentity':'${userAssignedIdentityResourceId}'}}"

```

This configuration will apply to all references for the app.

## Reference syntax

A Key Vault reference is of the form `@Microsoft.KeyVault({referenceString})`, where `{referenceString}` is replaced by one of the following options:

REFERENCE STRING	DESCRIPTION
<code>SecretUri=secretUri</code>	The <code>SecretUri</code> should be the full data-plane URI of a secret in Key Vault, optionally including a version, e.g., <code>https://myvault.vault.azure.net/secrets/mysecret/</code> or <code>https://myvault.vault.azure.net/secrets/mysecret/ec96f02080254f109c51a1f</code>
<code>VaultName=vaultName;SecretName=secretName;SecretVersion=secretVersion</code>	The <code>VaultName</code> is required and should be the name of your Key Vault resource. The <code>SecretName</code> is required and should be the name of the target secret. The <code>SecretVersion</code> is optional but if present indicates the version of the secret to use.

For example, a complete reference would look like the following:

```
@Microsoft.KeyVault(SecretUri=https://myvault.vault.azure.net/secrets/mysecret/)
```

Alternatively:

```
@Microsoft.KeyVault(VaultName=myvault;SecretName=mysecret)
```

## Rotation

If a version is not specified in the reference, then the app will use the latest version that exists in the key vault. When newer versions become available, such as with a rotation event, the app will automatically update and begin using the latest version within 24 hours. The delay is because App Service caches the values of the key vault references and refetches it every 24 hours. Any configuration changes to the app causes an immediate refetch of all referenced secrets.

## Source Application Settings from Key Vault

Key Vault references can be used as values for [Application Settings](#), allowing you to keep secrets in Key Vault instead of the site config. Application Settings are securely encrypted at rest, but if you need secret management capabilities, they should go into Key Vault.

To use a Key Vault reference for an [application setting](#), set the reference as the value of the setting. Your app can reference the secret through its key as normal. No code changes are required.

### TIP

Most application settings using Key Vault references should be marked as slot settings, as you should have separate vaults for each environment.

### Considerations for Azure Files mounting

Apps can use the `WEBSITE_CONTENTAZUREFILECONNECTIONSTRING` application setting to mount Azure Files as the file system. This setting has additional validation checks to ensure that the app can be properly started. The platform relies on having a content share within Azure Files, and it assumes a default name unless one is specified via the `WEBSITE_CONTENTSHARE` setting. For any requests which modify these settings, the platform will attempt to

validate if this content share exists, and it will attempt to create it if not. If it cannot locate or create the content share, the request is blocked.

When using Key Vault references for this setting, this validation check will fail by default, as the secret itself cannot be resolved while processing the incoming request. To avoid this issue, you can skip the validation by setting `WEBSITE_SKIP_CONTENTSHARE_VALIDATION` to "1". This will bypass all checks, and the content share will not be created for you. You should ensure it is created in advance.

**Caution**

If you skip validation and either the connection string or content share are invalid, the app will be unable to start properly and will only serve HTTP 500 errors.

As part of creating the site, it is also possible that attempted mounting of the content share could fail due to managed identity permissions not being propagated or the virtual network integration not being set up. You can defer setting up Azure Files until later in the deployment template to accommodate this. See [Azure Resource Manager deployment](#) to learn more. App Service will use a default file system until Azure Files is set up, and files are not copied over, so you will need to ensure that no deployment attempts occur during the interim period before Azure Files is mounted.

### Azure Resource Manager deployment

When automating resource deployments through Azure Resource Manager templates, you may need to sequence your dependencies in a particular order to make this feature work. Of note, you will need to define your application settings as their own resource, rather than using a `siteConfig` property in the site definition. This is because the site needs to be defined first so that the system-assigned identity is created with it and can be used in the access policy.

An example pseudo-template for a function app might look like the following:

```
{
 //...
 "resources": [
 {
 "type": "Microsoft.Storage/storageAccounts",
 "name": "[variables('storageAccountName')]",
 //...
 },
 {
 "type": "Microsoft.Insights/components",
 "name": "[variables('appInsightsName')]",
 //...
 },
 {
 "type": "Microsoft.Web/sites",
 "name": "[variables('functionAppName')]",
 "identity": {
 "type": "SystemAssigned"
 },
 //...
 },
 {
 "type": "config",
 "name": "appsettings",
 //...
 "dependsOn": [
 "[resourceId('Microsoft.Web/sites', variables('functionAppName'))]",
 "[resourceId('Microsoft.KeyVault/vaults', variables('keyVaultName'))]",
 "[resourceId('Microsoft.KeyVault/vaults/secrets', variables('keyVaultName'), variables('storageConnectionStringName'))]",
 "[resourceId('Microsoft.KeyVault/vaults/secrets', variables('keyVaultName'), variables('appInsightsKeyName'))]"
],
 "properties": {
 "AzureWebJobsStorage": "[concat('@Microsoft.KeyVault(SecretUri=',
 reference(variables('storageConnectionStringResourceId')).secretUriWithVersion, ')')]",
 "WEBSITE_CONTENTAZUREFILECONNECTIONSTRING": "[concat('@Microsoft.KeyVault(SecretUri=',
 reference(variables('storageConnectionStringResourceId')).secretUriWithVersion, ')')]",
 "APPINSIGHTS_INSTRUMENTATIONKEY": "[concat('@Microsoft.KeyVault(SecretUri=',
 reference(variables('appInsightsKeyId')).secretUriWithVersion, ')')]",
 "WEBSITE_ENABLE_SYNC_UPDATE_SITE": "true"
 }
 },
 {
 }
```

```

 "type": "sourcecontrols",
 "name": "web",
 //...
 "dependsOn": [
 "[resourceId('Microsoft.Web/sites', variables('functionAppName'))]",
 "[resourceId('Microsoft.Web/sites/config', variables('functionAppName')),
'appsettings')]"
],
 },
 {
 "type": "Microsoft.KeyVault/vaults",
 "name": "[variables('keyVaultName')]",
 //...
 "dependsOn": [
 "[resourceId('Microsoft.Web/sites', variables('functionAppName'))]"
],
 "properties": {
 //...
 "accessPolicies": [
 {
 "tenantId": "[reference(resourceId('Microsoft.Web/sites/',
variables('functionAppName')), '2020-12-01', 'Full').identity.tenantId]",
 "objectId": "[reference(resourceId('Microsoft.Web/sites/',
variables('functionAppName')), '2020-12-01', 'Full').identity.principalId]",
 "permissions": {
 "secrets": ["get"]
 }
 }
],
 "resources": [
 {
 "type": "secrets",
 "name": "[variables('storageConnectionStringName')]",
 //...
 "dependsOn": [
 "[resourceId('Microsoft.KeyVault/vaults', variables('keyVaultName'))]",
 "[resourceId('Microsoft.Storage/storageAccounts', variables('storageAccountName'))]"
],
 "properties": {
 "value": "[concat('DefaultEndpointsProtocol=https;AccountName=', variables('storageAccountName'), ';AccountKey=', listKeys(variables('storageAccountResourceId'), '2019-09-01').key1)]"
 }
 },
 {
 "type": "secrets",
 "name": "[variables('appInsightsKeyName')]",
 //...
 "dependsOn": [
 "[resourceId('Microsoft.KeyVault/vaults', variables('keyVaultName'))]",
 "[resourceId('Microsoft.Insights/components', variables('appInsightsName'))]"
],
 "properties": {
 "value": "[reference(resourceId('microsoft.insights/components',
variables('appInsightsName')), '2019-09-01').InstrumentationKey]"
 }
 }
]
 }
 }
}

```

#### NOTE

In this example, the source control deployment depends on the application settings. This is normally unsafe behavior, as the app setting update behaves asynchronously. However, because we have included the `WEBSITE_ENABLE_SYNC_UPDATE_SITE` application setting, the update is synchronous. This means that the source control deployment will only begin once the application settings have been fully updated. For more app settings, see [Environment variables and app settings in Azure App Service](#).

## Troubleshooting Key Vault References

If a reference is not resolved properly, the reference value will be used instead. This means that for application settings, an environment variable would be created whose value has the `@Microsoft.KeyVault(...)` syntax. This may cause the application to throw errors, as it was expecting a secret of a certain structure.

Most commonly, this is due to a misconfiguration of the [Key Vault access policy](#). However, it could also be due to a secret no longer existing or a syntax error in the reference itself.

If the syntax is correct, you can view other causes for error by checking the current resolution status in the portal. Navigate to Application Settings and select "Edit" for the reference in question. Below the setting configuration, you should see status information, including any errors. The absence of these implies that the reference syntax is invalid.

You can also use one of the built-in detectors to get additional information.

#### **Using the detector for App Service**

1. In the portal, navigate to your app.
2. Select **Diagnose and solve problems**.
3. Choose **Availability and Performance** and select **Web app down**.
4. Find **Key Vault Application Settings Diagnostics** and click **More info**.

#### **Using the detector for Azure Functions**

1. In the portal, navigate to your app.
2. Navigate to **Platform features**.
3. Select **Diagnose and solve problems**.
4. Choose **Availability and Performance** and select **Function app down or reporting errors**.
5. Click on **Key Vault Application Settings Diagnostics**.

# Use a TLS/SSL certificate in your code in Azure App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

In your application code, you can access the [public or private certificates you add to App Service](#). Your app code may act as a client and access an external service that requires certificate authentication, or it may need to perform cryptographic tasks. This how-to guide shows how to use public or private certificates in your application code.

This approach to using certificates in your code makes use of the TLS functionality in App Service, which requires your app to be in **Basic** tier or above. If your app is in **Free** or **Shared** tier, you can [include the certificate file in your app repository](#).

When you let App Service manage your TLS/SSL certificates, you can maintain the certificates and your application code separately and safeguard your sensitive data.

## Prerequisites

To follow this how-to guide:

- [Create an App Service app](#)
- [Add a certificate to your app](#)

## Find the thumbprint

In the [Azure portal](#), from the left menu, select **App Services > <app-name>**.

From the left navigation of your app, select **TLS/SSL settings**, then select **Private Key Certificates (.pfx)** or **Public Key Certificates (.cer)**.

Find the certificate you want to use and copy the thumbprint.

Private Key Certificates					
Status Filter					
All	Healthy	Warning	Expired	Expiration	Thumbprint
 Healthy	www.contoso.com			4/11/2020	6A3BCCA5CC4B0158F0A097CE9F39...

## Make the certificate accessible

To access a certificate in your app code, add its thumbprint to the `WEBSITE_LOAD_CERTIFICATES` app setting, by running the following command in the [Cloud Shell](#):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings WEBSITE_LOAD_CERTIFICATES=<comma-separated-certificate-thumbprints>
```

To make all your certificates accessible, set the value to `*`.

## Load certificate in Windows apps

The `WEBSITE_LOAD_CERTIFICATES` app setting makes the specified certificates accessible to your Windows hosted app in the Windows certificate store, in [Current User\My](#).

In C# code, you access the certificate by the certificate thumbprint. The following code loads a certificate with the thumbprint `E661583E8FABEF4C0BEF694CBC41C28FB81CD870`.

```
using System;
using System.Linq;
using System.Security.Cryptography.X509Certificates;

string certThumbprint = "E661583E8FABEF4C0BEF694CBC41C28FB81CD870";
bool validOnly = false;

using (X509Store certStore = new X509Store(StoreName.My, StoreLocation.CurrentUser))
{
 certStore.Open(OpenFlags.ReadOnly);

 X509Certificate2Collection certCollection = certStore.Certificates.Find(
 X509FindType.FindByThumbprint,
 // Replace below with your certificate's thumbprint
 certThumbprint,
 validOnly);

 // Get the first cert with the thumbprint
 X509Certificate2 cert = certCollection.OfType<X509Certificate2>().FirstOrDefault();

 if (cert is null)
 throw new Exception($"Certificate with thumbprint {certThumbprint} was not found");

 // Use certificate
 Console.WriteLine(cert.FriendlyName);

 // Consider to call Dispose() on the certificate after it's being used, available in .NET 4.6 and later
}
```

In Java code, you access the certificate from the "Windows-MY" store using the Subject Common Name field (see [Public key certificate](#)). The following code shows how to load a private key certificate:

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.RequestMapping;
import java.security.KeyStore;
import java.security.cert.Certificate;
import java.security.PrivateKey;

...
KeyStore ks = KeyStore.getInstance("Windows-MY");
ks.load(null, null);
Certificate cert = ks.getCertificate("<subject-cn>");
PrivateKey privKey = (PrivateKey) ks.getKey("<subject-cn>", ("<password>").toCharArray());

// Use the certificate and key
...
```

For languages that don't support or offer insufficient support for the Windows certificate store, see [Load certificate from file](#).

## Load certificate from file

If you need to load a certificate file that you upload manually, it's better to upload the certificate using [FTPS](#) instead of [Git](#), for example. You should keep sensitive data like a private certificate out of source control.

#### NOTE

ASP.NET and ASP.NET Core on Windows must access the certificate store even if you load a certificate from a file. To load a certificate file in a Windows .NET app, load the current user profile with the following command in the [Cloud Shell](#):

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
WEBSITE_LOAD_USER_PROFILE=1
```

This approach to using certificates in your code makes use of the TLS functionality in App Service, which requires your app to be in **Basic** tier or above.

The following C# example loads a public certificate from a relative path in your app:

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;

...
var bytes = File.ReadAllBytes("~/<relative-path-to-cert-file>");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate
```

To see how to load a TLS/SSL certificate from a file in Node.js, PHP, Python, Java, or Ruby, see the documentation for the respective language or web platform.

## Load certificate in Linux/Windows containers

The `WEBSITE_LOAD_CERTIFICATES` app settings makes the specified certificates accessible to your Windows or Linux container apps (including built-in Linux containers) as files. The files are found under the following directories:

CONTAINER PLATFORM	PUBLIC CERTIFICATES	PRIVATE CERTIFICATES
Windows container	<code>C:\appservice\certificates\public</code>	<code>C:\appservice\certificates\private</code>
Linux container	<code>/var/ssl/certs</code>	<code>/var/ssl/private</code>

The certificate file names are the certificate thumbprints.

#### NOTE

App Service injects the certificate paths into Windows containers as the following environment variables

`WEBSITE_PRIVATE_CERTS_PATH`, `WEBSITE_INTERMEDIATE_CERTS_PATH`, `WEBSITE_PUBLIC_CERTS_PATH`, and `WEBSITE_ROOT_CERTS_PATH`. It's better to reference the certificate path with the environment variables instead of hardcoding the certificate path, in case the certificate paths change in the future.

In addition, [Windows Server Core containers](#) load the certificates into the certificate store automatically, in `LocalMachine\My`. To load the certificates, follow the same pattern as [Load certificate in Windows apps](#). For Windows Nano based containers, use the file paths provided above to [Load the certificate directly from file](#).

The following C# code shows how to load a public certificate in a Linux app.

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;

...
var bytes = File.ReadAllBytes("/var/ssl/certs/<thumbprint>.der");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate
```

The following C# code shows how to load a private certificate in a Linux app.

```
using System;
using System.IO;
using System.Security.Cryptography.X509Certificates;
...
var bytes = File.ReadAllBytes("/var/ssl/private/<thumbprint>.p12");
var cert = new X509Certificate2(bytes);

// Use the loaded certificate
```

To see how to load a TLS/SSL certificate from a file in Node.js, PHP, Python, Java, or Ruby, see the documentation for the respective language or web platform.

## More resources

- [Secure a custom DNS name with a TLS/SSL binding in Azure App Service](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [FAQ : App Service Certificates](#)
- [Environment variables and app settings reference](#)

# Configure TLS mutual authentication for Azure App Service

11/2/2021 • 8 minutes to read • [Edit Online](#)

You can restrict access to your Azure App Service app by enabling different types of authentication for it. One way to do it is to request a client certificate when the client request is over TLS/SSL and validate the certificate. This mechanism is called TLS mutual authentication or client certificate authentication. This article shows how to set up your app to use client certificate authentication.

## NOTE

If you access your site over HTTP and not HTTPS, you will not receive any client certificate. So if your application requires client certificates, you should not allow requests to your application over HTTP.

## Prepare your web app

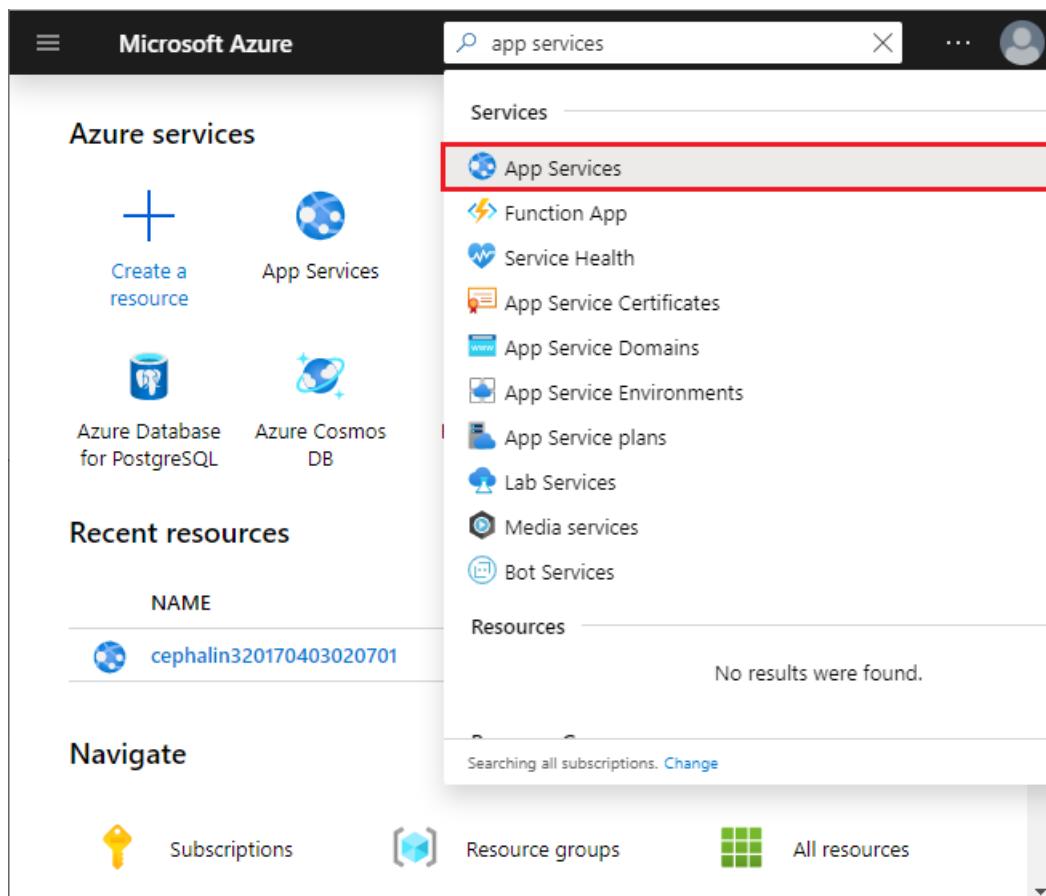
To create custom TLS/SSL bindings or enable client certificates for your App Service app, your [App Service plan](#) must be in the **Basic**, **Standard**, **Premium**, or **Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

### Sign in to Azure

Open the [Azure portal](#).

### Navigate to your web app

Search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar with the text "app services". Below the search bar, the "Services" section is expanded, showing various service icons and names. The "App Services" icon (a blue square with a white gear) is highlighted with a red box. Other listed services include Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. To the left of the search results, there's a sidebar titled "Azure services" with options like "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". Below the search results, there's a "Recent resources" section with a table showing a single item: "cephalin320170403020701". At the bottom, there's a "Navigate" section with links for "Subscriptions", "Resource groups", and "All resources".

On the App Services page, select the name of your web app.

The screenshot shows the Azure App Services management interface. At the top, there's a navigation bar with 'Home > App Services'. Below it, a header bar includes 'App Services' (with a Microsoft logo), 'Documentation', and various action buttons like '+ Add', 'Edit columns', 'Refresh', 'Assign tags', 'Start', 'Restart', and 'More'. A message 'Subscriptions: All 2 selected - Don't see a subscription? Open Directory + Subscription settings' is displayed. Below this are several filter buttons: 'Filter by na...', 'All subsc... ▾', 'All resou... ▾', 'All locati... ▾', 'All tags ▾', and 'No group... ▾'. A table lists '6 items':

Name	Status	App Type	App Service
cephalin320170403020701	Running	Web App	test-sku
denniseastusbot	Running	Web App	z76-z763
myFirstAzureWebApp20190...	Running	Web App	ServicePl...
WebApplicationASPDotNET...	Running	Web App	ServicePl...

You have landed on the management page of your web app.

#### Check the pricing tier

In the left-hand navigation of your web app page, scroll to the **Settings** section and select **Scale up (App Service plan)**.

The screenshot shows the left sidebar of a web app's settings page. At the top, it displays the app name 'cephalin320170403020701' and 'App Service'. Below is a search bar labeled 'Search (Ctrl+ /)'. The sidebar has a tree-like structure with the following items:

- Deployment Center
- Settings
  - Configuration
  - Container settings
  - Authentication / Authorization
  - Application Insights
  - Identity
  - Backups
  - Custom domains
  - TLS/SSL settings
  - Networking
- Scale up (App Service plan)

The 'Scale up (App Service plan)' item is highlighted with a red box.

Check to make sure that your web app is not in the F1 or D1 tier. Your web app's current tier is highlighted by a dark blue box.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

### Recommended pricing tiers

**F1**

Shared infrastructure  
1 GB memory  
60 minutes/day compute  
Free

**D1**

Shared infrastructure  
1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

### Included hardware

Every instance of your App Service plan will include the following hardware configuration:

#### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...

#### Memory

Memory available to run applications

Custom SSL is not supported in the F1 or D1 tier. If you need to scale up, follow the steps in the next section. Otherwise, close the Scale up page and skip the [Scale up your App Service plan](#) section.

#### Scale up your App Service plan

Select any of the non-free tiers (B1, B2, B3, or any tier in the Production category). For additional options, click See additional options.

Click **Apply**.



## Dev / Test

For less demanding workloads



## Production

For most production workloads



## Isolated

Advanced networking and scale

## Recommended pricing tiers

Shared infrastructure

**F1**

1 GB memory  
60 minutes/day compute  
Free

Shared infrastructure

**D1**

1 GB memory  
240 minutes/day compute  
<price>/Month (Estimated)

**B1**

100 total ACU  
1.75 GB memory  
A-Series compute equivalent  
<price>/Month (Estimated)

▼ See additional options

## Included features

Every app hosted on this App Service plan will have access to these features:



### Custom domains / SSL

Configure and purchase custom domains with SNI SSL bindings



### Manual scale

Up to 3 instances. Subject to availability.

## Included hardware

Every instance of your App Service plan will include the following hardware configuration:



### Azure Compute Units (ACU)

Dedicated compute resources used to run applications deployed in the App...



### Memory

Memory per instance available to run applications deployed and running in...

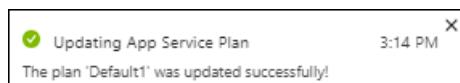


### Storage

10 GB disk storage shared by all apps deployed in the App Service plan.

**Apply**

When you see the following notification, the scale operation is complete.



## Enable client certificates

To set up your app to require client certificates:

- From the left navigation of your app's management page, select Configuration > General Settings.
- Set Client certificate mode to **Require**. Click **Save** at the top of the page.

To do the same with Azure CLI, run the following command in the [Cloud Shell](#):

```
az webapp update --set clientCertEnabled=true --name <app-name> --resource-group <group-name>
```

## Exclude paths from requiring authentication

When you enable mutual auth for your application, all paths under the root of your app require a client certificate for access. To remove this requirement for certain paths, define exclusion paths as part of your application configuration.

1. From the left navigation of your app's management page, select **Configuration > General Settings**.
2. Next to **Client exclusion paths**, click the edit icon.
3. Click **New path**, specify a path, or a list of paths separated by `,` or `;`, and click **OK**.
4. Click **Save** at the top of the page.

In the following screenshot, any path for your app that starts with `/public` does not request a client certificate. Path matching is case-insensitive.

The screenshot shows the Azure App Service configuration interface for an app named 'my-demo-app'. The 'General settings' tab is active. In the 'Incoming client certificates' section, the 'Client certificate mode' dropdown is set to 'Require' (which is highlighted with a red box). Below it, the 'Certificate exclusion paths' input field also contains a value ('/public') that is highlighted with a red box. The left sidebar lists various configuration categories like Overview, Activity log, and Deployment.

## Access client certificate

In App Service, TLS termination of the request happens at the frontend load balancer. When forwarding the request to your app code with [client certificates enabled](#), App Service injects an `X-ARR-ClientCert` request header with the client certificate. App Service does not do anything with this client certificate other than

forwarding it to your app. Your app code is responsible for validating the client certificate.

For ASP.NET, the client certificate is available through the `HttpRequest.ClientCertificate` property.

For other application stacks (Node.js, PHP etc.), the client cert is available in your app through a base64 encoded value in the `X-ARR-ClientCert` request header.

## ASP.NET 5+, ASP.NET Core 3.1 sample

For ASP.NET Core, middleware is provided to parse forwarded certificates. Separate middleware is provided to use the forwarded protocol headers. Both must be present for forwarded certificates to be accepted. You can place custom certificate validation logic in the [CertificateAuthentication options](#).

```

public class Startup
{
 public Startup(IConfiguration configuration)
 {
 Configuration = configuration;
 }

 public IConfiguration Configuration { get; }

 public void ConfigureServices(IServiceCollection services)
 {
 services.AddControllersWithViews();
 // Configure the application to use the protocol and client ip address forwarded by the frontend load
 balancer
 services.Configure<ForwardedHeadersOptions>(options =>
 {
 options.ForwardedHeaders =
 ForwardedHeaders.XForwardedFor | ForwardedHeaders.XForwardedProto;
 });

 // Configure the application to client certificate forwarded the frontend load balancer
 services.AddCertificateForwarding(options => { options.CertificateHeader = "X-ARR-ClientCert"; });

 // Add certificate authentication so when authorization is performed the user will be created from
 the certificate
 services.AddAuthentication(CertificateAuthenticationDefaults.AuthenticationScheme).AddCertificate();
 }

 public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
 {
 if (env.IsDevelopment())
 {
 app.UseDeveloperExceptionPage();
 }
 else
 {
 app.UseExceptionHandler("/Home/Error");
 app.UseHsts();
 }

 app.UseForwardedHeaders();
 app.UseCertificateForwarding();
 app.UseHttpsRedirection();

 app.UseAuthentication()
 app.UseAuthorization();

 app.UseStaticFiles();

 app.UseRouting();

 app.UseEndpoints(endpoints =>
 {
 endpoints.MapControllerRoute(
 name: "default",
 pattern: "{controller=Home}/{action=Index}/{id?}");
 });
 }
}

```

## ASP.NET WebForms sample

```

using System;
using System.Collections.Specialized;
using System.Security.Cryptography.X509Certificates;

```

```

using System.Web;

namespace ClientCertificateUsageSample
{
 public partial class Cert : System.Web.UI.Page
 {
 public string certHeader = "";
 public string errorString = "";
 private X509Certificate2 certificate = null;
 public string certThumbprint = "";
 public string certSubject = "";
 public string certIssuer = "";
 public string certSignatureAlg = "";
 public string certIssueDate = "";
 public string certExpiryDate = "";
 public bool isValidCert = false;

 //
 // Read the certificate from the header into an X509Certificate2 object
 // Display properties of the certificate on the page
 //
 protected void Page_Load(object sender, EventArgs e)
 {
 NameValueCollection headers = base.Request.Headers;
 certHeader = headers["X-ARR-ClientCert"];
 if (!String.IsNullOrEmpty(certHeader))
 {
 try
 {
 byte[] clientCertBytes = Convert.FromBase64String(certHeader);
 certificate = new X509Certificate2(clientCertBytes);
 certSubject = certificate.Subject;
 certIssuer = certificate.Issuer;
 certThumbprint = certificate.Thumbprint;
 certSignatureAlg = certificate.SignatureAlgorithm.FriendlyName;
 certIssueDate = certificate.NotBefore.ToShortDateString() + " " +
certificate.NotBefore.ToShortTimeString();
 certExpiryDate = certificate.NotAfter.ToShortDateString() + " " +
certificate.NotAfter.ToShortTimeString();
 }
 catch (Exception ex)
 {
 errorString = ex.ToString();
 }
 finally
 {
 isValidCert = IsValidClientCertificate();
 if (!isValidCert) Response.StatusCode = 403;
 else Response.StatusCode = 200;
 }
 }
 else
 {
 certHeader = "";
 }
 }

 //
 // This is a SAMPLE verification routine. Depending on your application logic and security
 requirements,
 // you should modify this method
 //
 private bool IsValidClientCertificate()
 {
 // In this example we will only accept the certificate as a valid certificate if all the
 conditions below are met:
 // 1. The certificate is not expired and is active for the current time on server.
 // 2. The subject name of the certificate has the common name nildevecc
 // 3. The issuer name of the certificate has the common name nildevecc and organization name

```

```

Microsoft Corp
 // 4. The thumbprint of the certificate is 30757A2E831977D8BD9C8496E4C99AB26CB9622B
 //
 // This example does NOT test that this certificate is chained to a Trusted Root Authority
 (or revoked) on the server
 // and it allows for self signed certificates
 //

 if (certificate == null || !String.IsNullOrEmpty(errorString)) return false;

 // 1. Check time validity of certificate
 if (DateTime.Compare(DateTime.Now, certificate.NotBefore) < 0 ||
 DateTime.Compare(DateTime.Now, certificate.NotAfter) > 0) return false;

 // 2. Check subject name of certificate
 bool foundSubject = false;
 string[] certSubjectData = certificate.Subject.Split(new char[] { ',' },
 StringSplitOptions.RemoveEmptyEntries);
 foreach (string s in certSubjectData)
 {
 if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
 {
 foundSubject = true;
 break;
 }
 }
 if (!foundSubject) return false;

 // 3. Check issuer name of certificate
 bool foundIssuerCN = false, foundIssuerO = false;
 string[] certIssuerData = certificate.Issuer.Split(new char[] { ',' },
 StringSplitOptions.RemoveEmptyEntries);
 foreach (string s in certIssuerData)
 {
 if (String.Compare(s.Trim(), "CN=nildevecc") == 0)
 {
 foundIssuerCN = true;
 if (foundIssuerO) break;
 }

 if (String.Compare(s.Trim(), "O=Microsoft Corp") == 0)
 {
 foundIssuerO = true;
 if (foundIssuerCN) break;
 }
 }

 if (!foundIssuerCN || !foundIssuerO) return false;

 // 4. Check thumprint of certificate
 if (String.Compare(certificate.Thumbprint.Trim().ToUpper(),
 "30757A2E831977D8BD9C8496E4C99AB26CB9622B") != 0) return false;

 return true;
}
}
}

```

## Node.js sample

The following Node.js sample code gets the `x-ARR-ClientCert` header and uses [node-forge](#) to convert the base64-encoded PEM string into a certificate object and validate it:

```

import { NextFunction, Request, Response } from 'express';
import { pki, md, asn1 } from 'node-forge';

export class AuthorizationHandler {
 public static authorizeClientCertificate(req: Request, res: Response, next: NextFunction): void {
 try {
 // Get header
 const header = req.get('X-ARR-ClientCert');
 if (!header) throw new Error('UNAUTHORIZED');

 // Convert from PEM to pki.CERT
 const pem = `-----BEGIN CERTIFICATE-----${header}-----END CERTIFICATE-----`;
 const incomingCert: pki.Certificate = pki.certificateFromPem(pem);

 // Validate certificate thumbprint
 const fingerPrint =
 md.sha1.create().update(asn1.toDer(pki.certificateToAsn1(incomingCert)).getBytes()).digest().toHex();
 if (fingerPrint.toLowerCase() !== 'abcdef1234567890abcdef1234567890abcdef12') throw new
 Error('UNAUTHORIZED');

 // Validate time validity
 const currentDate = new Date();
 if (currentDate < incomingCert.validity.notBefore || currentDate >
 incomingCert.validity.notAfter) throw new Error('UNAUTHORIZED');

 // Validate issuer
 if (incomingCert.issuer.hash.toLowerCase() !== 'abcdef1234567890abcdef1234567890abcdef12') throw
 new Error('UNAUTHORIZED');

 // Validate subject
 if (incomingCert.subject.hash.toLowerCase() !== 'abcdef1234567890abcdef1234567890abcdef12') throw
 new Error('UNAUTHORIZED');

 next();
 } catch (e) {
 if (e instanceof Error && e.message === 'UNAUTHORIZED') {
 res.status(401).send();
 } else {
 next(e);
 }
 }
 }
}

```

## Java sample

The following Java class encodes the certificate from `X-ARR-ClientCert` to an `X509Certificate` instance. `certificateIsValid()` validates that the certificate's thumbprint matches the one given in the constructor and that certificate has not expired.

```

import java.io.ByteArrayInputStream;
import java.security.NoSuchAlgorithmException;
import java.security.cert.*;
import java.security.MessageDigest;

import sun.security.provider.X509Factory;

import javax.xml.bind.DatatypeConverter;
import java.util.Base64;
import java.util.Date;

public class ClientCertValidator {

 private String thumbprint;

```

```

private X509Certificate certificate;

/**
 * Constructor.
 * @param certificate The certificate from the "X-ARR-ClientCert" HTTP header
 * @param thumbprint The thumbprint to check against
 * @throws CertificateException If the certificate factory cannot be created.
 */
public ClientCertValidator(String certificate, String thumbprint) throws CertificateException {
 certificate = certificate
 .replaceAll(X509Factory.BEGIN_CERT, "")
 .replaceAll(X509Factory.END_CERT, "");
 CertificateFactory cf = CertificateFactory.getInstance("X.509");
 byte [] base64Bytes = Base64.getDecoder().decode(certificate);
 X509Certificate X509cert = (X509Certificate) cf.generateCertificate(new
ByteArrayInputStream(base64Bytes));

 this.setCertificate(X509cert);
 this.setThumbprint(thumbprint);
}

/**
 * Check that the certificate's thumbprint matches the one given in the constructor, and that the
 * certificate has not expired.
 * @return True if the certificate's thumbprint matches and has not expired. False otherwise.
 */
public boolean certificateIsValid() throws NoSuchAlgorithmException, CertificateEncodingException {
 return certificateHasNotExpired() && thumbprintIsValid();
}

/**
 * Check certificate's timestamp.
 * @return Returns true if the certificate has not expired. Returns false if it has expired.
 */
private boolean certificateHasNotExpired() {
 Date currentTime = new java.util.Date();
 try {
 this.getCertificate().checkValidity(currentTime);
 } catch (CertificateExpiredException | CertificateNotYetValidException e) {
 return false;
 }
 return true;
}

/**
 * Check the certificate's thumbprint matches the given one.
 * @return Returns true if the thumbprints match. False otherwise.
 */
private boolean thumbprintIsValid() throws NoSuchAlgorithmException, CertificateEncodingException {
 MessageDigest md = MessageDigest.getInstance("SHA-1");
 byte[] der = this.getCertificate().getEncoded();
 md.update(der);
 byte[] digest = md.digest();
 String digestHex = DatatypeConverter.printHexBinary(digest);
 return digestHex.toLowerCase().equals(this.getThumbprint().toLowerCase());
}

// Getters and setters

public void setThumbprint(String thumbprint) {
 this.thumbprint = thumbprint;
}

public String getThumbprint() {
 return this.thumbprint;
}

public X509Certificate getCertificate() {
 return certificate;
}

```

```
}

public void setCertificate(X509Certificate certificate) {
 this.certificate = certificate;
}

}
```

# Encryption at rest using customer-managed keys

11/2/2021 • 4 minutes to read • [Edit Online](#)

Encrypting your web app's application data at rest requires an Azure Storage Account and an Azure Key Vault. These services are used when you run your app from a deployment package.

- [Azure Storage provides encryption at rest](#). You can use system-provided keys or your own, customer-managed keys. This is where your application data is stored when it's not running in a web app in Azure.
- [Running from a deployment package](#) is a deployment feature of App Service. It allows you to deploy your site content from an Azure Storage Account using a Shared Access Signature (SAS) URL.
- [Key Vault references](#) are a security feature of App Service. It allows you to import secrets at runtime as application settings. Use this to encrypt the SAS URL of your Azure Storage Account.

## Set up encryption at rest

### Create an Azure Storage account

First, [create an Azure Storage account](#) and [encrypt it with customer-managed keys](#). Once the storage account is created, use the [Azure Storage Explorer](#) to upload package files.

Next, use the Storage Explorer to [generate an SAS](#).

#### NOTE

Save this SAS URL, this is used later to enable secure access of the deployment package at runtime.

### Configure running from a package from your storage account

Once you upload your file to Blob storage and have an SAS URL for the file, set the `WEBSITE_RUN_FROM_PACKAGE` application setting to the SAS URL. The following example does it by using Azure CLI:

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings
WEBSITE_RUN_FROM_PACKAGE="<your-SAS-URL>"
```

Adding this application setting causes your web app to restart. After the app has restarted, browse to it and make sure that the app has started correctly using the deployment package. If the application didn't start correctly, see the [Run from package troubleshooting guide](#).

### Encrypt the application setting using Key Vault references

Now you can replace the value of the `WEBSITE_RUN_FROM_PACKAGE` application setting with a Key Vault reference to the SAS-encoded URL. This keeps the SAS URL encrypted in Key Vault, which provides an extra layer of security.

1. Use the following `az keyvault create` command to create a Key Vault instance.

```
az keyvault create --name "Contoso-Vault" --resource-group <group-name> --location eastus
```

2. Follow [these instructions to grant your app access](#) to your key vault:

3. Use the following `az keyvault secret set` command to add your external URL as a secret in your key vault:

```
az keyvault secret set --vault-name "Contoso-Vault" --name "external-url" --value "<SAS-URL>"
```

4. Use the following `az webapp config appsettings set` command to create the `WEBSITE_RUN_FROM_PACKAGE` application setting with the value as a Key Vault reference to the external URL:

```
az webapp config appsettings set --settings
WEBSITE_RUN_FROM_PACKAGE="@Microsoft.KeyVault(SecretUri=https://Contoso-
Vault.vault.azure.net/secrets/external-url/<secret-version>")"
```

The `<secret-version>` will be in the output of the previous `az keyvault secret set` command.

Updating this application setting causes your web app to restart. After the app has restarted, browse to it make sure it has started correctly using the Key Vault reference.

## How to rotate the access token

It is best practice to periodically rotate the SAS key of your storage account. To ensure the web app does not inadvertently loose access, you must also update the SAS URL in Key Vault.

1. Rotate the SAS key by navigating to your storage account in the Azure portal. Under **Settings > Access keys**, click the icon to rotate the SAS key.
2. Copy the new SAS URL, and use the following command to set the updated SAS URL in your key vault:

```
az keyvault secret set --vault-name "Contoso-Vault" --name "external-url" --value "<SAS-URL>"
```

3. Update the key vault reference in your application setting to the new secret version:

```
az webapp config appsettings set --settings
WEBSITE_RUN_FROM_PACKAGE="@Microsoft.KeyVault(SecretUri=https://Contoso-
Vault.vault.azure.net/secrets/external-url/<secret-version>")"
```

The `<secret-version>` will be in the output of the previous `az keyvault secret set` command.

## How to revoke the web app's data access

There are two methods to revoke the web app's access to the storage account.

### Rotate the SAS key for the Azure Storage account

If the SAS key for the storage account is rotated, the web app will no longer have access to the storage account, but it will continue to run with the last downloaded version of the package file. Restart the web app to clear the last downloaded version.

### Remove the web app's access to Key Vault

You can revoke the web app's access to the site data by disabling the web app's access to Key Vault. To do this, remove the access policy for the web app's identity. This is the same identity you created earlier while configuring key vault references.

## Summary

Your application files are now encrypted at rest in your storage account. When your web app starts, it retrieves the SAS URL from your key vault. Finally, the web app loads the application files from the storage account.

If you need to revoke the web app's access to your storage account, you can either revoke access to the key vault or rotate the storage account keys, which invalidates the SAS URL.

## Frequently Asked Questions

### **Is there any additional charge for running my web app from the deployment package?**

Only the cost associated with the Azure Storage Account and any applicable egress charges.

### **How does running from the deployment package affect my web app?**

- Running your app from the deployment package makes `wwwroot/` read-only. Your app receives an error when it attempts to write to this directory.
- TAR and GZIP formats are not supported.
- This feature is not compatible with local cache.

## Next steps

- [Key Vault references for App Service](#)
- [Azure Storage encryption for data at rest](#)

# Scale up an app in Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to scale your app in Azure App Service. There are two workflows for scaling, scale up and scale out, and this article explains the scale up workflow.

- **Scale up:** Get more CPU, memory, disk space, and extra features like dedicated virtual machines (VMs), custom domains and certificates, staging slots, autoscaling, and more. You scale up by changing the pricing tier of the App Service plan that your app belongs to.
- **Scale out:** Increase the number of VM instances that run your app. You can scale out to as many as 30 instances, depending on your pricing tier. [App Service Environments](#) in **Isolated** tier further increases your scale-out count to 100 instances. For more information about scaling out, see [Scale instance count manually or automatically](#). There, you find out how to use autoscaling, which is to scale instance count automatically based on predefined rules and schedules.

The scale settings take only seconds to apply and affect all apps in your [App Service plan](#). They don't require you to change your code or redeploy your application.

For information about the pricing and features of individual App Service plans, see [App Service Pricing Details](#).

## NOTE

Before you switch an App Service plan from the **Free** tier, you must first remove the [spending limits](#) in place for your Azure subscription. To view or change options for your Microsoft Azure App Service subscription, see [Microsoft Azure Subscriptions](#).

## Scale up your pricing tier

### NOTE

To scale up to **PremiumV3** tier, see [Configure PremiumV3 tier for App Service](#).

1. In your browser, open the [Azure portal](#).
2. In your App Service app page, from the left menu, select **Scale Up (App Service plan)**.
3. Choose your tier, and then select **Apply**. Select the different categories (for example, **Production**) and also **See additional options** to show more tiers.

The screenshot shows the 'dotnetcore-back-end - Scale up (App Service plan)' configuration page. On the left, a sidebar lists various settings like Identity, Backups, Custom domains, TLS/SSL settings, Networking, and 'Scale up (App Service plan)' (which is highlighted with a red box). The main area displays 'Recommended pricing tiers' with three options: F1 (Free), D1 (9.67 USD/Month), and B1 (55.80 USD/Month). Below this, the 'Included features' section lists 'Custom domains' (Configure and purchase custom domain names). The 'Included hardware' section details 'Azure Compute Units (ACU)' (Dedicated compute resources used to run applications deployed in the App Service Plan), 'Memory' (Memory available to run applications deployed and running in the App Service plan), and 'Storage' (1 GB disk storage shared by all apps deployed in the App Service plan). At the bottom is a large blue 'Apply' button.

When the operation is complete, you see a notification pop-up with a green success check mark.

## Scale related resources

If your app depends on other services, such as Azure SQL Database or Azure Storage, you can scale up these resources separately. These resources aren't managed by the App Service plan.

1. In the **Overview** page for your app, select the **Resource group** link.

The screenshot shows the 'my-demo-app' Overview page. The left sidebar includes links for Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, and Deployment. The main area displays resource details: Resource group (change) myResourceGroup, Status Running, Location West Europe, Subscription (change) mySubscription, and Subscription ID 00000000-0000-0000-0000-000000000000. A red box highlights the 'Resource group (change)' link 'myResourceGroup'.

2. In the **Summary** part of the **Resource group** page, select a resource that you want to scale. The following screenshot shows a SQL Database resource.

The screenshot shows the Azure portal interface for a resource group named 'myResourceGroup'. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Deployments, Policies, Properties, Locks, and Export template. The 'Overview' tab is selected. At the top right, it shows 'Subscription (change) mySubscription', 'Subscription ID 00000000-0000-0000-0000-000000000000', 'Tags (change) Click here to add tags', 'Deployments 5 Succeeded', and a search bar. Below the search bar is a table with columns 'NAME' and 'TYPE'. There are four items listed: 'dotnetcoredb' (SQL server), 'coreDB (dotnetcoredb/coreDB)' (SQL database, highlighted with a red box), 'myAppServicePlan' (App Service plan), and 'my-demo-app' (App Service). A filter bar at the top of the table allows filtering by name, type, and location.

To scale up the related resource, see the documentation for the specific resource type. For example, to scale up a single SQL Database, see [Scale single database resources in Azure SQL Database](#). To scale up a Azure Database for MySQL resource, see [Scale MySQL resources](#).

## Compare pricing tiers

For detailed information, such as VM sizes for each pricing tier, see [App Service Pricing Details](#).

For a table of service limits, quotas, and constraints, and supported features in each tier, see [App Service limits](#).

## More resources

[Scale instance count manually or automatically](#)

[Configure PremiumV3 tier for App Service](#)

# Configure PremiumV3 tier for Azure App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

The new **PremiumV3** pricing tier gives you faster processors, SSD storage, and quadruple the memory-to-core ratio of the existing pricing tiers (double the **PremiumV2** tier). With the performance advantage, you could save money by running your apps on fewer instances. In this article, you learn how to create an app in **PremiumV3** tier or scale up an app to **PremiumV3** tier.

## Prerequisites

To scale-up an app to **PremiumV3**, you need to have an Azure App Service app that runs in a pricing tier lower than **PremiumV3**, and the app must be running in an App Service deployment that supports **PremiumV3**.

## PremiumV3 availability

The **PremiumV3** tier is available for both native and container apps, including both Windows containers and Linux containers.

### NOTE

Any Windows containers running in the **Premium Container** tier during the preview period continue to function as is, but the **Premium Container** tier will continue to remain in preview. The **PremiumV3** tier is the official replacement for the **Premium Container** tier.

**PremiumV3** is available in some Azure regions and availability in additional regions is being added continually. To see if it's available in your region, run the following Azure CLI command in the [Azure Cloud Shell](#):

```
az appservice list-locations --sku P1V3
```

## Create an app in PremiumV3 tier

The pricing tier of an App Service app is defined in the [App Service plan](#) that it runs on. You can create an App Service plan by itself or as part of app creation.

When configuring the App Service plan in the [Azure portal](#), select **Pricing tier**.

Select **Production**, then select **P1V3**, **P2V3**, or **P3V3**, then click **Apply**.


**Dev / Test**  
For less demanding workloads


**Production**  
For most production workloads


**Isolated**  
Advanced networking and scale

**Recommended pricing tiers**

<b>S1</b> 100 total ACU 1.75 GB memory A-Series compute equivalent 73.00 USD/Month (Estimated)	<b>P1V2</b> 210 total ACU 3.5 GB memory Dv2-Series compute equivalent 146.00 USD/Month (Estimated)	<b>P2V2</b> 420 total ACU 7 GB memory Dv2-Series compute equivalent 292.00 USD/Month (Estimated)
<b>P3V2</b> 840 total ACU 14 GB memory Dv2-Series compute equivalent 584.00 USD/Month (Estimated)	<b>P1V3</b> 195 minimum ACU/vCPU 8 GB memory 2 vCPU /Month (Estimated)	<b>P2V3</b> 195 minimum ACU/vCPU 16 GB memory 4 vCPU /Month (Estimated)
<b>P3V3</b> 195 minimum ACU/vCPU 32 GB memory 8 vCPU /Month (Estimated)		

[▼ See additional options](#)

**Included features**  
Every app hosted on this App Service plan will have access to these features:

 <b>Custom domains / SSL</b> Configure and purchase custom domains with SNI and IP SSL bindings	 <b>Auto scale</b> Up to 30 instances. Subject to availability.	 <b>Staging slots</b> Up to 20 staging slots to use for testing and deployments before swapping them into production.	 <b>Daily backups</b> Backup your app 50 times daily.
 <b>Traffic manager</b> Improve performance and availability by routing traffic between multiple instances of your app.			

[Apply](#)

**Included hardware**  
Every instance of your App Service plan will include the following hardware configuration:

 <b>Azure Compute Units (ACU)</b> Dedicated compute resources used to run applications deployed in the App Service Plan. <a href="#">Learn more</a>	 <b>Memory</b> Memory per instance available to run applications deployed and running in the App Service plan.	 <b>Storage</b> 250 GB disk storage shared by all apps deployed in the App Service plan.
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

### IMPORTANT

If you don't see P1V3, P2V3, and P3V3 as options, or if the options are greyed out, then **PremiumV3** likely isn't available in the underlying App Service deployment that contains the App Service plan. See [Scale up from an unsupported resource group and region combination](#) for more details.

## Scale up an existing app to PremiumV3 tier

Before scaling an existing app to PremiumV3 tier, make sure that **PremiumV3** is available. For information, see [PremiumV3 availability](#). If it's not available, see [Scale up from an unsupported resource group and region combination](#).

Depending on your hosting environment, scaling up may require extra steps.

In the [Azure portal](#), open your App Service app page.

In the left navigation of your App Service app page, select **Scale up (App Service plan)**.

The screenshot shows the Azure portal interface for an App Service named 'my-demo-app'. On the left, there's a sidebar with various settings like Identity, Backups, Custom domains, TLS/SSL settings, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'Scale up (App Service plan)' option is highlighted with a red box. The main pane shows a message about .NET Framework 4.8 deployment. Below that, under the 'Essentials' section, there are options for Resource group, Status, Location, Subscription, and Subscription ID.

Select **Production**, then select P1V3, P2V3, or P3V3, then click **Apply**.

This screenshot shows the 'Recommended pricing tiers' section of the Azure App Service Plan configuration page. It lists several tiers: S1, P1V2, P2V2, P3V2, P1V3, P2V3, and P3V3. The 'Production' tier (P1V3) is highlighted with a red box. The P1V3 tier details are: 195 minimum ACU/vCPU, 8 GB memory, 2 vCPU, and 146.00 USD/Month (Estimated). A link 'See additional options' is visible at the bottom.

## Included features

Every app hosted on this App Service plan will have access to these features:

- Custom domains / SSL**  
Configure and purchase custom domains with SNI and IP SSL bindings
- Auto scale**  
Up to 30 instances. Subject to availability.
- Staging slots**  
Up to 20 staging slots to use for testing and deployments before swapping them into production.
- Daily backups**  
Backup your app 50 times daily.
- Traffic manager**  
Improve performance and availability by routing traffic between multiple instances of your app.

## Included hardware

Every instance of your App Service plan will include the following hardware configuration:

- Azure Compute Units (ACU)**  
Dedicated compute resources used to run applications deployed in the App Service Plan. [Learn more](#)
- Memory**  
Memory per instance available to run applications deployed and running in the App Service plan.
- Storage**  
250 GB disk storage shared by all apps deployed in the App Service plan.

**Apply**

If your operation finishes successfully, your app's overview page shows that it's now in a PremiumV3 tier.

<a href="#">Essentials</a>	
Resource group ( <a href="#">change</a> )	URL
<a href="#">myResourceGroup</a>	<a href="https://my-demo-app.azurewebsites.net">https://my-demo-app.azurewebsites.net</a>
Status	App Service Plan
Running	<a href="#">myAppServicePlan (P1v3: 1)</a>
Location	FTP/deployment username
West Central US	my-demo-app\deployuser
Subscription ( <a href="#">change</a> )	FTP hostname
<a href="#">MySubscription</a>	ftp://waws-prod-cy4-009.ftp.azurewebsites.wind...
Subscription ID	FTPS hostname
00000000-0000-0000-0000-000000000000	ftps://waws-prod-cy4-009.ftp.azurewebsites.wind...
Tags ( <a href="#">change</a> )	
<a href="#">Click here to add tags</a>	

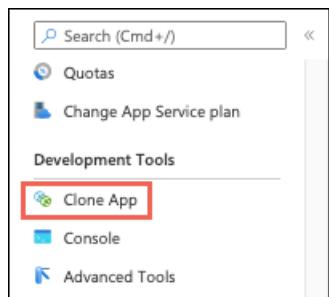
### If you get an error

Some App Service plans can't scale up to the PremiumV3 tier if the underlying App Service deployment doesn't support PremiumV3. See [Scale up from an unsupported resource group and region combination](#) for more details.

## Scale up from an unsupported resource group and region combination

If your app runs in an App Service deployment where **PremiumV3** isn't available, or if your app runs in a region that currently does not support **PremiumV3**, you need to re-deploy your app to take advantage of **PremiumV3**. You have two options:

- Create an app in a new resource group and with a new App Service plan. When creating the App Service plan, select a **PremiumV3** tier. This step ensures that the App Service plan is deployed into a deployment unit that supports **PremiumV3**. Then, redeploy your application code into the newly created app. Even if you scale the App Service plan down to a lower tier to save costs, you can always scale back up to **PremiumV3** because the deployment unit supports it.
- If your app already runs in an existing **Premium** tier, then you can clone your app with all app settings, connection strings, and deployment configuration into a new resource group on a new app service plan that uses **PremiumV3**.



In the **Clone app** page, you can create an App Service plan using **PremiumV3** in the region you want, and specify the app settings and configuration that you want to clone.

## Moving from Premium Container to Premium V3 SKU

If you have an app which is using the preview Premium Container SKU and you would like to move to the new Premium V3 SKU, you need to redeploy your app to take advantage of **PremiumV3**. To do this, see the first option in [Scale up from an unsupported resource group and region combination](#)

## Automate with scripts

You can automate app creation in the **PremiumV3** tier with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

## Azure CLI

The following command creates an App Service plan in *P1V3*. You can run it in the Cloud Shell. The options for `--sku` are *P1V3*, *P2V3*, and *P3V3*.

```
az appservice plan create \
--resource-group <resource_group_name> \
--name <app_service_plan_name> \
--sku P1V3
```

## Azure PowerShell

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

The following command creates an App Service plan in *P1V3*. The options for `-WorkerSize` are *Small*, *Medium*, and *Large*.

```
New-AzAppServicePlan -ResourceGroupName <resource_group_name> `
-Name <app_service_plan_name> `
-Location <region_name> `
-Tier "PremiumV3" `
-WorkerSize "Small"
```

## More resources

[Scale up an app in Azure Scale instance count manually or automatically](#)

# Get started with Autoscale in Azure

11/2/2021 • 5 minutes to read • [Edit Online](#)

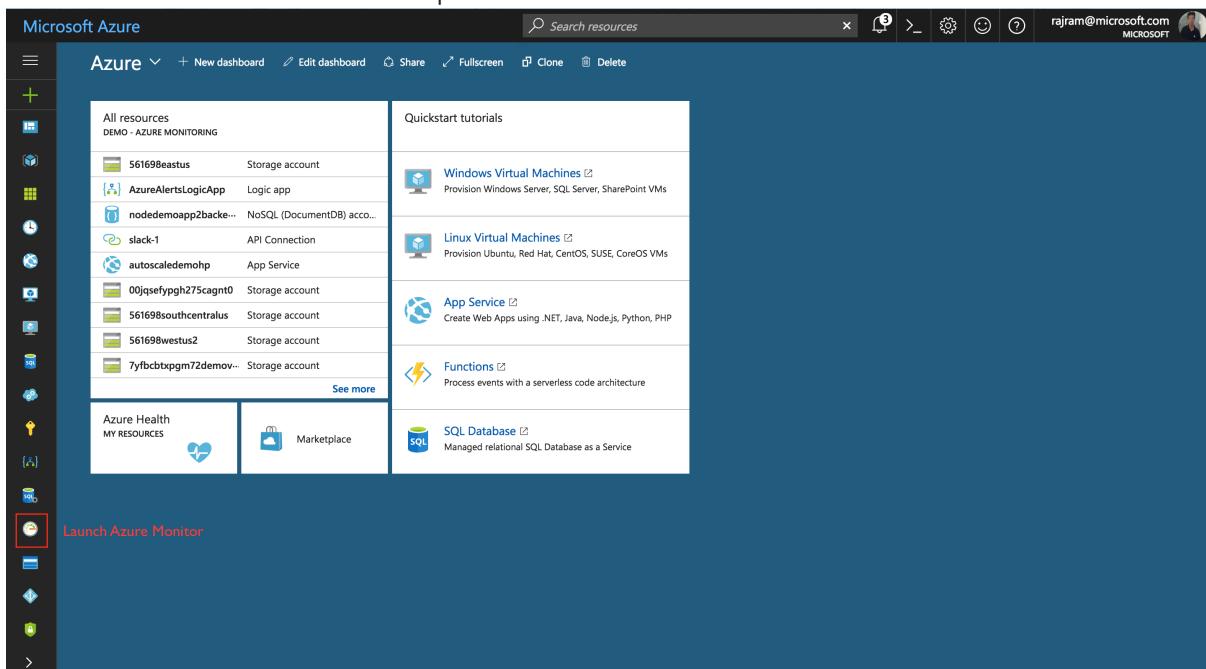
This article describes how to set up your Autoscale settings for your resource in the Microsoft Azure portal.

Azure Monitor autoscale applies only to [Virtual Machine scale sets](#), [Cloud Services](#), [App Service - Web Apps](#), and [API Management services](#).

## Discover the Autoscale settings in your subscription

You can discover all the resources for which Autoscale is applicable in Azure Monitor. Use the following steps for a step-by-step walkthrough:

1. Open the [Azure portal](#).
2. Click the Azure Monitor icon in the left pane.



3. Click **Autoscale** to view all the resources for which Autoscale is applicable, along with their current Autoscale status.

The screenshot shows the Microsoft Azure Monitor - Autoscale blade. On the left, there's a navigation pane with sections like Explore, Manage, and Health. Under Manage, the 'Autoscale' option is selected. The main area displays a table of resources with columns: NAME, RESOURCE TYPE, RESOURCE GROUP, LOCATION, INSTANCE COUNT, and AUTOSCALE STATUS. The table lists various resources such as WebWorkerDemo (Cloud service), demovmss (Virtual machine scale set), and several App Service plans. Most resources have an instance count of 1, except for demovmss, CPUBasedScaleAsp, HolidaySpikeAsp, staticscaleasp, WeekdayTrafficAsp, contoso-mvc-app-asp, and contoso-web-api-asp which have 2 instances each. The 'Enabled' status is indicated by a green circle with a checkmark.

You can use the filter pane at the top to scope down the list to select resources in a specific resource group, specific resource types, or a specific resource.

For each resource, you will find the current instance count and the Autoscale status. The Autoscale status can be:

- **Not configured:** You have not enabled Autoscale yet for this resource.
- **Enabled:** You have enabled Autoscale for this resource.
- **Disabled:** You have disabled Autoscale for this resource.

## Create your first Autoscale setting

Let's now go through a simple step-by-step walkthrough to create your first Autoscale setting.

1. Open the **Autoscale** blade in Azure Monitor and select a resource that you want to scale. (The following steps use an App Service plan associated with a web app. You can [create your first ASP.NET web app in Azure in 5 minutes](#).)
2. Note that the current instance count is 1. Click **Enable autoscale**.

The screenshot shows the 'Autoscale setting' blade for the 'nodedemompp2sp' App Service plan. At the top, there are buttons for Save, Discard, and Disable autoscale. Below that, there's a 'Configure' button and links for Run history, JSON, and Notify. A section titled 'Override condition' contains a slider for 'Instance count' which is currently set to 1. A note below says 'Your autoscale configuration is disabled. To reinstate your configuration, enable autoscale.' At the bottom, there's a prominent blue 'Enable autoscale' button.

3. Provide a name for the scale setting, and then click **Add a rule**. Notice the scale rule options that open as a context pane on the right side. By default, this sets the option to scale your instance count by 1 if the CPU percentage of the resource exceeds 70 percent. Leave it at its default values and click **Add**.

The screenshot shows the 'Autoscale setting' configuration for the 'nodeapp2sp' App Service plan. A scale rule is defined with the following criteria:

- Metric name:** CPU Percentage
- Time aggregation:** Average
- Operator:** Greater than
- Threshold:** 70
- Duration (in minutes):** 10
- Action:** Increase count by 1

4. You've now created your first scale rule. Note that the UX recommends best practices and states that "It is recommended to have at least one scale in rule." To do so:

- Click **Add a rule**.
- Set **Operator** to **Less than**.
- Set **Threshold** to **20**.
- Set **Operation** to **Decrease count by**.

You should now have a scale setting that scales out/scales in based on CPU usage.

The screenshot shows the 'Autoscale setting' configuration for the 'nodeapp2sp' App Service plan with two rules:

- Scale out:** When **nodedemoapp2sp** (Average) CpuPercentage > 70, Increase instance count by 1
- Scale in:** When **nodedemoapp2sp** (Average) CpuPercentage < 20, Decrease instance count by 1

5. Click **Save**.

Congratulations! You've now successfully created your first scale setting to autoscale your web app based on CPU usage.

#### NOTE

The same steps are applicable to get started with a virtual machine scale set or cloud service role.

# Other considerations

## Scale based on a schedule

In addition to scale based on CPU, you can set your scale differently for specific days of the week.

1. Click **Add a scale condition**.
2. Setting the scale mode and the rules is the same as the default condition.
3. Select **Repeat specific days** for the schedule.
4. Select the days and the start/end time for when the scale condition should be applied.

Microsoft Azure Monitor - Autoscale > Autoscale setting

Autoscale setting  
nodedemoapp2sp (App Service plan)

Save Discard Disable autoscale

Rules

Scale in

When nodedemoapp2sp (Average) CpuPercentage < 20 Decrease instance count by 1

+ Add a rule

Instance limits Minimum 2 Maximum 5 Default 2

Schedule This scale condition is executed when none of the other scale condition(s) match

Auto created scale condition

Scale mode Scale based on a metric Scale to a specific instance count (selected)

Instance count 1

Schedule Specify start/end dates Repeat specific days (selected)

Repeat every Monday Tuesday Wednesday Thursday Friday Saturday Sunday (checked)

Timezone (UTC-08:00) Pacific Time (US & Canada)

Start time 00:00

End time 11:59

+ Add a scale condition

## Scale differently on specific dates

In addition to scale based on CPU, you can set your scale differently for specific dates.

1. Click **Add a scale condition**.
2. Setting the scale mode and the rules is the same as the default condition.
3. Select **Specify start/end dates** for the schedule.
4. Select the start/end dates and the start/end time for when the scale condition should be applied.

## View the scale history of your resource

Whenever your resource is scaled up or down, an event is logged in the activity log. You can view the scale history of your resource for the past 24 hours by switching to the **Run history** tab.

OPERATION NAME	STATUS	EVENT CATEGO...	TIME	TIME STAMP	SUBSCRIPTION	EVENT INITIATED BY	RESOURCE TYPE	RESOURCE
Scaleup	Succeeded	Autoscale	14 h ago	Sun May 07 2...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	14 h ago	Sat May 06 2...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Sat May 06 2...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	2 d ago	Fri May 05 20...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	4 d ago	Wed May 03 ...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaledown	Succeeded	Autoscale	4 d ago	Wed May 03 ...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp
Scaleup	Succeeded	Autoscale	4 d ago	Wed May 03 ...	Demo - Azure Monitoring	Microsoft.insights/autoscale...	Microsoft.Web/serverFarms	serverFarms/contoso-mvc-app-asp

If you want to view the complete scale history (for up to 90 days), select [Click here to see more details](#). The activity log opens, with Autoscale pre-selected for your resource and category.

## View the scale definition of your resource

Autoscale is an Azure Resource Manager resource. You can view the scale definition in JSON by switching to the **JSON** tab.

The screenshot shows the Microsoft Azure portal interface for managing an Autoscale setting. The top navigation bar includes 'Microsoft Azure', 'Monitor - Autoscale > Autoscale setting', 'Search resources', and a user profile for 'rajram@microsoft.com'. Below the navigation is a toolbar with 'Save', 'Discard', and 'Disable autoscale' buttons. A sidebar on the left contains icons for various Azure services. The main content area has tabs for 'Configure', 'Run history', 'JSON', and 'Notify', with 'JSON' selected. The 'Autoscale setting name' is set to 'contoso-mvc-autoscale'. The 'Autoscale setting resource id' and 'Target resource id' are also specified. The JSON code shown defines two scale conditions: 'Build conference scale condition' and 'Weekend scale condition', each with specific capacity rules and fixed date ranges.

```
4 "type": "Microsoft.Insights/autoscaleSettings",
5 "location": "West US 2",
6 "tags": {
7 "$type": "Microsoft.WindowsAzure.Management.Common.Storage.CasePreservedDictionary, Microsoft.WindowsAzure.Management.Common.Storage"
8 },
9 "properties": {
10 "profiles": [
11 {
12 "name": "Build conference scale condition",
13 "capacity": {
14 "minimum": "10",
15 "maximum": "10",
16 "default": "10"
17 },
18 "rules": [],
19 "fixedDate": {
20 "timeZone": "Pacific Standard Time",
21 "start": "2017-05-07T00:00:00Z",
22 "end": "2017-05-13T23:59:00Z"
23 }
24 },
25 {
26 "name": "Weekend scale condition",
27 "capacity": {
28 "minimum": "1",
29 "maximum": "1",
30 "default": "1"
31 },
32 "rules": []
33 }
34]
35 }
36 }
```

You can make changes in JSON directly, if required. These changes will be reflected after you save them.

## Disable Autoscale and manually scale your instances

There might be times when you want to disable your current scale setting and manually scale your resource.

Click the **Disable autoscale** button at the top.

The screenshot shows the Microsoft Azure portal with the 'Disable autoscale' button highlighted in blue. A confirmation dialog box is displayed, asking 'Disable autoscale? Disable your current autoscale configuration? You can reinstate your configuration anytime.' with 'Yes' and 'No' buttons. Below the dialog, the 'Scale mode' section is visible, showing 'Default' and 'Normal scale behavior1' with a link to edit it. It also shows the 'Scale mode' dropdown set to 'Scale based on a metric'.

### NOTE

This option disables your configuration. However, you can get back to it after you enable Autoscale again.

You can now set the number of instances that you want to scale to manually.

The screenshot shows the Microsoft Azure portal with the 'Override condition' section expanded. It features a slider for 'Instance count' set to 4, with a text input field showing the value '4'. Below the slider, a message states 'Your autoscale configuration is disabled. To reinstate your configuration, enable autoscale.' At the bottom is a blue 'Enable autoscale' button.

You can always return to Autoscale by clicking **Enable autoscale** and then **Save**.

## Cool-down period effects

Autoscale uses a cool-down period to prevent "flapping", which is the rapid, repetitive up and down scaling of

instances. For more information, see [Autoscale evaluation steps](#). Other valuable information on flapping and understanding how to monitor the autoscale engine can be found in [Autoscale Best Practices](#) and [Troubleshooting autoscale](#) respectively.

## Route traffic to healthy instances (App Service)

When your Azure web app is scaled out to multiple instances, App Service can perform health checks on your instances to route traffic to the healthy instances. To learn more, see [this article on App Service Health check](#).

## Moving Autoscale to a different region

This section describes how to move Azure autoscale to another region under the same Subscription, and Resource Group. You can use REST API to move autoscale settings.

### Prerequisite

1. Ensure that the subscription and Resource Group are available and the details in both the source and destination regions are identical.
2. Ensure that Azure autoscale is available in the [Azure region you want to move to](#).

### Move

Use [REST API](#) to create an autoscale setting in the new environment. The autoscale setting created in the destination region will be a copy of the autoscale setting in the source region.

[Diagnostic settings](#) that were created in association with the autoscale setting in the source region cannot be moved. You will need to recreate diagnostic settings in the destination region, after the creation of autoscale settings is completed.

### Learn more about moving resources across Azure regions

To learn more about moving resources between regions and disaster recovery in Azure, refer to [Move resources to a new resource group or subscription](#)

## Next steps

- [Create an Activity Log Alert to monitor all Autoscale engine operations on your subscription](#)
- [Create an Activity Log Alert to monitor all failed Autoscale scale-in/scale-out operations on your subscription](#)

# High-density hosting on Azure App Service using per-app scaling

11/2/2021 • 3 minutes to read • [Edit Online](#)

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

When using App Service, you can scale your apps by scaling the [App Service plan](#) they run on. When multiple apps are run in the same App Service plan, each scaled-out instance runs all the apps in the plan.

*Per-app scaling* can be enabled at the App Service plan level to allow for scaling an app independently from the App Service plan that hosts it. This way, an App Service plan can be scaled to 10 instances, but an app can be set to use only five.

## NOTE

Per-app scaling is available only for **Standard**, **Premium**, **Premium V2**, **Premium V3**, and **Isolated** pricing tiers.

Apps are allocated to available App Service plan using a best effort approach for an even distribution across instances. While an even distribution is not guaranteed, the platform will make sure that two instances of the same app will not be hosted on the same App Service plan instance.

The platform does not rely on metrics to decide on worker allocation. Applications are rebalanced only when instances are added or removed from the App Service plan.

## Per app scaling using PowerShell

Create a plan with per-app scaling by passing in the `-PerSiteScaling $true` parameter to the `New-AzAppServicePlan` cmdlet.

```
New-AzAppServicePlan -ResourceGroupName $ResourceGroup -Name $AppServicePlan `
 -Location $Location `
 -Tier Premium -WorkerSize Small `
 -NumberofWorkers 5 -PerSiteScaling $true
```

Enable per-app scaling with an existing App Service Plan by passing in the `-PerSiteScaling $true` parameter to the `Set-AzAppServicePlan` cmdlet.

```
Enable per-app scaling for the App Service Plan using the "PerSiteScaling" parameter.
Set-AzAppServicePlan -ResourceGroupName $ResourceGroup `
 -Name $AppServicePlan -PerSiteScaling $true
```

At the app level, configure the number of instances the app can use in the App Service plan.

In the example below, the app is limited to two instances regardless of how many instances the underlying app service plan scales out to.

```
Get the app we want to configure to use "PerSiteScaling"
$newapp = Get-AzWebApp -ResourceGroupName $ResourceGroup -Name $webapp

Modify the NumberOfWorkers setting to the desired value.
$newapp.SiteConfig.NumberOfWorkers = 2

Post updated app back to azure
Set-AzWebApp $newapp
```

#### IMPORTANT

`$newapp.SiteConfig.NumberOfWorkers` is different from `$newapp.MaxNumberOfWorkers`. Per-app scaling uses `$newapp.SiteConfig.NumberOfWorkers` to determine the scale characteristics of the app.

## Per-app scaling using Azure Resource Manager

The following Azure Resource Manager template creates:

- An App Service plan that's scaled out to 10 instances
- an app that's configured to scale to a max of five instances.

The App Service plan is setting the **PerSiteScaling** property to true `"perSiteScaling": true`. The app is setting the **number of workers** to use to 5 `"properties": { "numberOfWorkers": "5" }`.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
 "appServicePlanName": { "type": "string" },
 "appName": { "type": "string" }
 },
 "resources": [
 {
 "comments": "App Service Plan with per site perSiteScaling = true",
 "type": "Microsoft.Web/serverFarms",
 "sku": {
 "name": "P1",
 "tier": "Premium",
 "size": "P1",
 "family": "P",
 "capacity": 10
 },
 "name": "[parameters('appServicePlanName')]",
 "apiVersion": "2015-08-01",
 "location": "West US",
 "properties": {
 "name": "[parameters('appServicePlanName')]",
 "perSiteScaling": true
 }
 },
 {
 "type": "Microsoft.Web/sites",
 "name": "[parameters('appName')]",
 "apiVersion": "2015-08-01-preview",
 "location": "West US",
 "dependsOn": ["[resourceId('Microsoft.Web/serverFarms', parameters('appServicePlanName'))]"],
 "properties": { "serverFarmId": "[resourceId('Microsoft.Web/serverFarms', parameters('appServicePlanName'))]" },
 "resources": [{
 "comments": "",
 "type": "config",
 "name": "web",
 "apiVersion": "2015-08-01",
 "location": "West US",
 "dependsOn": ["[resourceId('Microsoft.Web/Sites', parameters('appName'))]"],
 "properties": { "numberOfWorkers": "5" }
 }]
 }
]
}
```

## Recommended configuration for high-density hosting

Per app scaling is a feature that is enabled in both global Azure regions and [App Service Environments](#).

However, the recommended strategy is to use App Service Environments to take advantage of their advanced features and the larger App Service plan capacity.

Follow these steps to configure high-density hosting for your apps:

1. Designate an App Service plan as the high-density plan and scale it out to the desired capacity.
  2. Set the `PerSiteScaling` flag to true on the App Service plan.
  3. New apps are created and assigned to that App Service plan with the `numberOfWorkers` property set to 1.
    - Using this configuration yields the highest density possible.
  4. The number of workers can be configured independently per app to grant additional resources as needed.
- For example:
- A high-use app can set `numberOfWorkers` to 3 to have more processing capacity for that app.

- Low-use apps would set **numberOfWorkers** to 1.

## Next steps

- [Azure App Service plans in-depth overview](#)
- [Introduction to App Service Environment](#)

# Monitoring App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

When you have critical applications and business processes relying on Azure resources, you want to monitor those resources for their availability, performance, and operation. This article describes the monitoring data generated by App Service and shipped to [Azure Monitor](#). You can also use [built-in diagnostics to monitor resources](#) to assist with debugging an App Service app. If you're unfamiliar with the features of Azure Monitor common to all Azure services, read [Monitoring Azure resources with Azure Monitor](#).

## Monitoring data

App Service collects the same kinds of monitoring data as other Azure resources that are described in [Monitoring data from Azure resources](#).

See [Monitoring App Service data reference](#) for detailed information on App Service metrics and logs.

App Service also provides built-in diagnostics to assist with debugging apps. See [Enable diagnostics logging](#) for more information on enabling the built-in logs. To monitor App Service instances, see [Monitor App Service instances using Health check](#).

## Collection and routing

Platform metrics and the Activity log are collected and stored automatically, but can be routed to other locations by using a diagnostic setting.

Resource Logs aren't collected and stored until you create a diagnostic setting and route them to one or more locations.

See [Create diagnostic setting to collect platform logs and metrics in Azure](#) for the detailed process for creating a diagnostic setting using the Azure portal, CLI, or PowerShell. When you create a diagnostic setting, you specify which categories of logs to collect. The categories for *App Service* are listed in [App Service monitoring data reference](#).

The metrics and logs you can collect are discussed in the following sections.

## Analyzing metrics

You can analyze metrics for *App Service* with metrics from other Azure services using metrics explorer by opening **Metrics** from the **Azure Monitor** menu. See [Getting started with Azure Metrics Explorer](#) for details on using this tool.

For a list of platform metrics collected for App Service, see [Monitoring App Service data reference metrics](#)

For reference, you can see a list of [all resource metrics supported in Azure Monitor](#).

## Analyzing logs

Data in Azure Monitor Logs is stored in tables where each table has its own set of unique properties.

All resource logs in Azure Monitor have the same fields followed by service-specific fields. The common schema is outlined in [Azure Monitor resource log schema](#).

The [Activity log](#) is a type of platform log that provides insight into subscription-level events. You can view it

independently or route to Azure Monitor Logs. Routing to Azure Monitor Logs gives the benefit of using Log Analytics to run complex queries.

For a list of types of resource logs collected for App Service, see [Monitoring App Service data reference](#)

For a list of queryable tables used by Azure Monitor Logs and Log Analytics, see [Monitoring App Service data reference](#).

## Sample Kusto queries

### IMPORTANT

When you select **Logs** from the App Service menu, Log Analytics is opened with the query scope set to the current resource. This means that log queries will only include data from that resource. If you want to run a query that includes data from other [resource] or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

The following sample query can help you monitor app logs using `AppServiceAppLogs`:

```
AppServiceAppLogs
| project CustomLevel, _ResourceId
| summarize count() by CustomLevel, _ResourceId
```

The following sample query can help you monitor HTTP logs using `AppServiceHTTPLogs` where the `HTTP response code` is `500` or higher:

```
AppServiceHTTPLogs
//| where ResourceId = "MyResourceId" // Uncomment to get results for a specific resource Id when querying
over a group of Apps
| where ScStatus >= 500
| reduce by strcat(CsMethod, ':\\\', CsUriStem)
```

The following sample query can help you monitor HTTP 500 errors by joining `AppServiceConsoleLogs` and `AppserviceHTTPLogs`:

```
let myHttp = AppServiceHTTPLogs | where ScStatus == 500 | project TimeGen=substring(TimeGenerated, 0, 19),
CsUriStem, ScStatus;

let myConsole = AppServiceConsoleLogs | project TimeGen=substring(TimeGenerated, 0, 19), ResultDescription;

myHttp | join myConsole on TimeGen | project TimeGen, CsUriStem, ScStatus, ResultDescription;
```

See [Azure Monitor queries for App Service](#) for more sample queries.

## Alerts

Azure Monitor alerts proactively notify you when important conditions are found in your monitoring data. They allow you to identify and address issues in your system before your customers notice them. You can set alerts on [metrics](#), [logs](#), and the [activity log](#).

If you're running an application on App Service [Azure Monitor Application Insights](#) offers more types of alerts.

The following table lists common and recommended alert rules for App Service.

ALERT TYPE	CONDITION	EXAMPLES
Metric	Average connections	When number of connections exceed a set value
Metric	HTTP 404	When HTTP 404 responses exceed a set value
Metric	HTTP Server Errors	When HTTP 5xx errors exceed a set value
Activity Log	Create or Update Web App	When app is created or updated
Activity Log	Delete Web App	When app is deleted
Activity Log	Restart Web App	When app is restarted
Activity Log	Stop Web App	When app is stopped

## Next steps

- See [Monitoring App Service data reference](#) for a reference of metrics, logs, and other important values created by App Service.
- See [Monitoring Azure resources with Azure Monitor](#) for details on monitoring Azure resources.

# Monitor apps in Azure App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

Azure App Service provides built-in monitoring functionality for web apps, mobile, and API apps in the [Azure portal](#).

In the Azure portal, you can review *quotas* and *metrics* for an app and App Service plan, and set up *alerts* and *autoscaling* rules based metrics.

## Understand quotas

Apps that are hosted in App Service are subject to certain limits on the resources they can use. The limits are defined by the App Service plan that's associated with the app.

### 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.

If the app is hosted in a *Free* or *Shared* plan, the limits on the resources that the app can use are defined by quotas.

If the app is hosted in a *Basic*, *Standard*, or *Premium* plan, the limits on the resources that they can use are set by the *size* (Small, Medium, Large) and *instance count* (1, 2, 3, and so on) of the App Service plan.

Quotas for Free or Shared apps are:

QUOTA	DESCRIPTION
CPU (Short)	The amount of CPU allowed for this app in a 5-minute interval. This quota resets every five minutes.
CPU (Day)	The total amount of CPU allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Memory	The total amount of memory allowed for this app.
Bandwidth	The total amount of outgoing bandwidth allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
Filesystem	The total amount of storage allowed.

The only quota applicable to apps that are hosted in *Basic*, *Standard*, and *Premium* is Filesystem.

For more information about the specific quotas, limits, and features available to the various App Service SKUs, see [Azure Subscription service limits](#).

### Quota enforcement

If an app exceeds the *CPU (short)*, *CPU (Day)*, or *bandwidth* quota, the app is stopped until the quota resets.

During this time, all incoming requests result in an HTTP 403 error.

# Error 403 - This web app is stopped.

The web app you have attempted to reach is currently stopped and does not accept any requests. Please try to reload the page or visit it again soon.

If you are the web app administrator, please find the common 403 error scenarios and resolution [here](#). For further troubleshooting tools and recommendations, please visit [Azure Portal](#).

If the app Memory quota is exceeded, the app is stopped temporarily.

If the Filesystem quota is exceeded, any write operation fails. Write operation failures include any writes to logs.

You can increase or remove quotas from your app by upgrading your App Service plan.

## Understand metrics

### NOTE

**File System Usage** is a new metric being rolled out globally, no data is expected unless your app is hosted in an App Service Environment.

### IMPORTANT

**Average Response Time** will be deprecated to avoid confusion with metric aggregations. Use **Response Time** as a replacement.

### NOTE

Metrics for an app include the requests to the app's SCM site(Kudu). This includes requests to view the site's logstream using Kudu. Logstream requests may span several minutes, which will affect the Request Time metrics. Users should be aware of this relationship when using these metrics with autoscale logic.

Metrics provide information about the app or the App Service plan's behavior.

For an app, the available metrics are:

METRIC	DESCRIPTION
<b>Response Time</b>	The time taken for the app to serve requests, in seconds.
<b>Average Response Time (deprecated)</b>	The average time taken for the app to serve requests, in seconds.

METRIC	DESCRIPTION
Average memory working set	The average amount of memory used by the app, in megabytes (MiB).
Connections	The number of bound sockets existing in the sandbox (w3wp.exe and its child processes). A bound socket is created by calling bind()/connect() APIs and remains until said socket is closed with CloseHandle()/closesocket().
CPU Time	The amount of CPU consumed by the app, in seconds. For more information about this metric, see <a href="#">CPU time vs CPU percentage</a> .
Current Assemblies	The current number of Assemblies loaded across all AppDomains in this application.
Data In	The amount of incoming bandwidth consumed by the app, in MiB.
Data Out	The amount of outgoing bandwidth consumed by the app, in MiB.
File System Usage	The amount of usage in bytes by storage share.
Gen 0 Garbage Collections	The number of times the generation 0 objects are garbage collected since the start of the app process. Higher generation GCs include all lower generation GCs.
Gen 1 Garbage Collections	The number of times the generation 1 objects are garbage collected since the start of the app process. Higher generation GCs include all lower generation GCs.
Gen 2 Garbage Collections	The number of times the generation 2 objects are garbage collected since the start of the app process.
Handle Count	The total number of handles currently open by the app process.
Health Check Status	The average health status across the application's instances in the App Service Plan.
Http 2xx	The count of requests resulting in an HTTP status code $\geq 200$ but $< 300$ .
Http 3xx	The count of requests resulting in an HTTP status code $\geq 300$ but $< 400$ .
Http 401	The count of requests resulting in HTTP 401 status code.
Http 403	The count of requests resulting in HTTP 403 status code.
Http 404	The count of requests resulting in HTTP 404 status code.
Http 406	The count of requests resulting in HTTP 406 status code.

METRIC	DESCRIPTION
Http 4xx	The count of requests resulting in an HTTP status code $\geq 400$ but $< 500$ .
Http Server Errors	The count of requests resulting in an HTTP status code $\geq 500$ but $< 600$ .
IO Other Bytes Per Second	The rate at which the app process is issuing bytes to I/O operations that don't involve data, such as control operations.
IO Other Operations Per Second	The rate at which the app process is issuing I/O operations that aren't read or write operations.
IO Read Bytes Per Second	The rate at which the app process is reading bytes from I/O operations.
IO Read Operations Per Second	The rate at which the app process is issuing read I/O operations.
IO Write Bytes Per Second	The rate at which the app process is writing bytes to I/O operations.
IO Write Operations Per Second	The rate at which the app process is issuing write I/O operations.
Memory working set	The current amount of memory used by the app, in MiB.
Private Bytes	Private Bytes is the current size, in bytes, of memory that the app process has allocated that can't be shared with other processes.
Requests	The total number of requests regardless of their resulting HTTP status code.
Requests In Application Queue	The number of requests in the application request queue.
Thread Count	The number of threads currently active in the app process.
Total App Domains	The current number of AppDomains loaded in this application.
Total App Domains Unloaded	The total number of AppDomains unloaded since the start of the application.

For an App Service plan, the available metrics are:

**NOTE**

App Service plan metrics are available only for plans in *Basic*, *Standard*, and *Premium* tiers.

METRIC	DESCRIPTION
CPU Percentage	The average CPU used across all instances of the plan.
Memory Percentage	The average memory used across all instances of the plan.
Data In	The average incoming bandwidth used across all instances of the plan.
Data Out	The average outgoing bandwidth used across all instances of the plan.
Disk Queue Length	The average number of both read and write requests that were queued on storage. A high disk queue length is an indication of an app that might be slowing down because of excessive disk I/O.
Http Queue Length	The average number of HTTP requests that had to sit on the queue before being fulfilled. A high or increasing HTTP Queue length is a symptom of a plan under heavy load.

### CPU time vs CPU percentage

There are two metrics that reflect CPU usage:

**CPU Time:** Useful for apps hosted in Free or Shared plans, because one of their quotas is defined in CPU minutes used by the app.

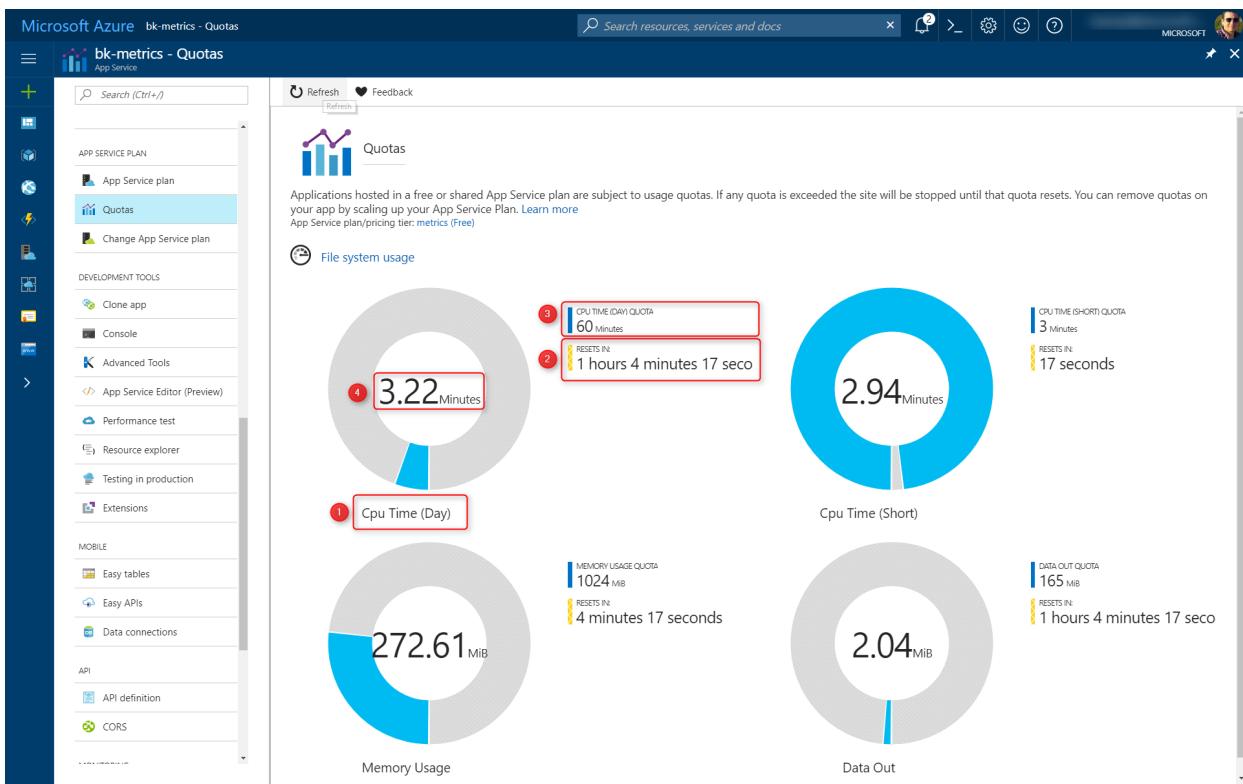
**CPU percentage:** Useful for apps hosted in Basic, Standard, and Premium plans, because they can be scaled out. CPU percentage is a good indication of the overall usage across all instances.

## Metrics granularity and retention policy

Metrics for an app and app service plan are logged and aggregated by the service and [retained according to these rules](#).

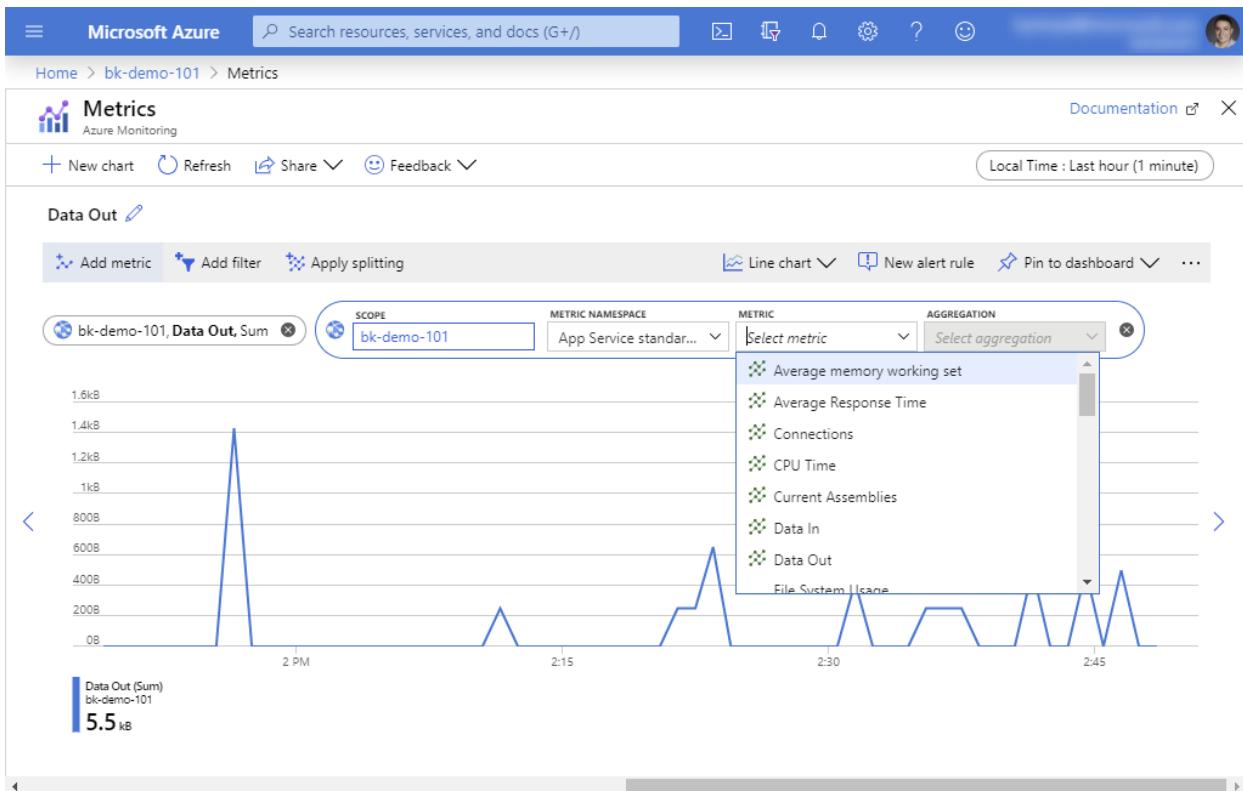
## Monitoring quotas and metrics in the Azure portal

To review the status of the various quotas and metrics that affect an app, go to the [Azure portal](#).



To find quotas, select **Settings > Quotas**. On the chart, you can review:

1. The quota name.
2. Its reset interval.
3. Its current limit.
4. Its current value.



You can access metrics directly from the resource **Overview** page. Here you'll see charts representing some of the apps metrics.

Clicking on any of those charts will take you to the metrics view where you can create custom charts, query different metrics and much more.

To learn more about metrics, see [Monitor service metrics](#).

## Alerts and autoscale

Metrics for an app or an App Service plan can be hooked up to alerts. For more information, see [Receive alert notifications](#).

App Service apps hosted in Basic or higher App Service plans support autoscale. With autoscale, you can configure rules that monitor the App Service plan metrics. Rules can increase or decrease the instance count, which can provide additional resources as needed. Rules can also help you save money when the app is over-provisioned.

For more information about autoscale, see [How to scale](#) and [Best practices for Azure Monitor autoscaling](#).

# Enable diagnostics logging for apps in Azure App Service

11/2/2021 • 9 minutes to read • [Edit Online](#)

## Overview

Azure provides built-in diagnostics to assist with debugging an [App Service app](#). In this article, you learn how to enable diagnostic logging and add instrumentation to your application, as well as how to access the information logged by Azure.

This article uses the [Azure portal](#) and Azure CLI to work with diagnostic logs. For information on working with diagnostic logs using Visual Studio, see [Troubleshooting Azure in Visual Studio](#).

### NOTE

In addition to the logging instructions in this article, there's new, integrated logging capability with Azure Monitoring. You'll find more on this capability in the [Send logs to Azure Monitor](#) section.

Type	Platform	Location	Description
Application logging	Windows, Linux	App Service file system and/or Azure Storage blobs	Logs messages generated by your application code. The messages can be generated by the web framework you choose, or from your application code directly using the standard logging pattern of your language. Each message is assigned one of the following categories: <b>Critical, Error, Warning, Info, Debug, and Trace</b> . You can select how verbose you want the logging to be by setting the severity level when you enable application logging.
Web server logging	Windows	App Service file system or Azure Storage blobs	Raw HTTP request data in the <a href="#">W3C extended log file format</a> . Each log message includes data such as the HTTP method, resource URI, client IP, client port, user agent, response code, and so on.

Type	Platform	Location	Description
Detailed Error Messages	Windows	App Service file system	Copies of the <i>.htm</i> error pages that would have been sent to the client browser. For security reasons, detailed error pages shouldn't be sent to clients in production, but App Service can save the error page each time an application error occurs that has HTTP code 400 or greater. The page may contain information that can help determine why the server returns the error code.
Failed request tracing	Windows	App Service file system	Detailed tracing information on failed requests, including a trace of the IIS components used to process the request and the time taken in each component. It's useful if you want to improve site performance or isolate a specific HTTP error. One folder is generated for each failed request, which contains the XML log file, and the XSL stylesheet to view the log file with.
Deployment logging	Windows, Linux	App Service file system	Logs for when you publish content to an app. Deployment logging happens automatically and there are no configurable settings for deployment logging. It helps you determine why a deployment failed. For example, if you use a <a href="#">custom deployment script</a> , you might use deployment logging to determine why the script is failing.

#### NOTE

App Service provides a dedicated, interactive diagnostics tool to help you troubleshoot your application. For more information, see [Azure App Service diagnostics overview](#).

In addition, you can use other Azure services to improve the logging and monitoring capabilities of your app, such as [Azure Monitor](#).

## Enable application logging (Windows)

**NOTE**

Application logging for blob storage can only use storage accounts in the same region as the App Service

To enable application logging for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

Select **On** for either **Application Logging (Filesystem)** or **Application Logging (Blob)**, or both.

The **Filesystem** option is for temporary debugging purposes, and turns itself off in 12 hours. The **Blob** option is for long-term logging, and needs a blob storage container to write logs to. The **Blob** option also includes additional information in the log messages, such as the ID of the origin VM instance of the log message (`InstanceId`), thread ID (`Tid`), and a more granular timestamp (`EventTickCount`).

**NOTE**

Currently only .NET application logs can be written to the blob storage. Java, PHP, Node.js, Python application logs can only be stored on the App Service file system (without code modifications to write logs to external storage).

Also, if you [regenerate your storage account's access keys](#), you must reset the respective logging configuration to use the updated access keys. To do this:

1. In the **Configure** tab, set the respective logging feature to **Off**. Save your setting.
2. Enable logging to the storage account blob again. Save your setting.

Select the **Level**, or the level of details to log. The following table shows the log categories included in each level:

LEVEL	INCLUDED CATEGORIES
Disabled	None
Error	Error, Critical
Warning	Warning, Error, Critical
Information	Info, Warning, Error, Critical
Verbose	Trace, Debug, Info, Warning, Error, Critical (all categories)

When finished, select **Save**.

**NOTE**

If you write logs to blobs, the retention policy no longer applies if you delete the app but keep the logs in the blobs. For more information, see [Costs that might accrue after resource deletion](#).

## Enable application logging (Linux/Container)

To enable application logging for Linux apps or custom container apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

In **Application logging**, select **File System**.

In **Quota (MB)**, specify the disk quota for the application logs. In **Retention Period (Days)**, set the number of days the logs should be retained.

When finished, select **Save**.

## Enable web server logging

To enable web server logging for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

For **Web server logging**, select **Storage** to store logs on blob storage, or **File System** to store logs on the App Service file system.

In **Retention Period (Days)**, set the number of days the logs should be retained.

### NOTE

If you [regenerate your storage account's access keys](#), you must reset the respective logging configuration to use the updated keys. To do this:

1. In the **Configure** tab, set the respective logging feature to **Off**. Save your setting.
2. Enable logging to the storage account blob again. Save your setting.

When finished, select **Save**.

### NOTE

If you write logs to blobs, the retention policy no longer applies if you delete the app but keep the logs in the blobs. For more information, see [Costs that might accrue after resource deletion](#).

## Log detailed errors

To save the error page or failed request tracing for Windows apps in the [Azure portal](#), navigate to your app and select **App Service logs**.

Under **Detailed Error Logging** or **Failed Request Tracing**, select **On**, then select **Save**.

Both types of logs are stored in the App Service file system. Up to 50 errors (files/folders) are retained. When the number of HTML files exceeds 50, the oldest error files are automatically deleted.

The Failed Request Tracing feature by default captures a log of requests that failed with HTTP status codes between 400 and 600. To specify custom rules, you can override the `<traceFailedRequests>` section in the `web.config` file.

## Add log messages in code

In your application code, you use the usual logging facilities to send log messages to the application logs. For example:

- ASP.NET applications can use the [System.Diagnostics.Trace](#) class to log information to the application diagnostics log. For example:

```
System.Diagnostics.Trace.TraceError("If you're seeing this, something bad happened");
```

- By default, ASP.NET Core uses the [Microsoft.Extensions.Logging.AzureAppServices](#) logging provider. For

more information, see [ASP.NET Core logging in Azure](#). For information about WebJobs SDK logging, see [Get started with the Azure WebJobs SDK](#)

## Stream logs

Before you stream logs in real time, enable the log type that you want. Any information written to files ending in .txt, .log, or .htm that are stored in the `/LogFiles` directory (`d:/home/logfiles`) is streamed by App Service.

### NOTE

Some types of logging buffer write to the log file, which can result in out of order events in the stream. For example, an application log entry that occurs when a user visits a page may be displayed in the stream before the corresponding HTTP log entry for the page request.

### In Azure portal

To stream logs in the [Azure portal](#), navigate to your app and select **Log stream**.

### In Cloud Shell

To stream logs live in [Cloud Shell](#), use the following command:

### IMPORTANT

This command may not work with web apps hosted in a Linux app service plan.

```
az webapp log tail --name appname --resource-group myResourceGroup
```

To filter specific log types, such as HTTP, use the **--Provider** parameter. For example:

```
az webapp log tail --name appname --resource-group myResourceGroup --provider http
```

### In local terminal

To stream logs in the local console, [install Azure CLI](#) and [sign in to your account](#). Once signed in, follow the [instructions for Cloud Shell](#)

## Access log files

If you configure the Azure Storage blobs option for a log type, you need a client tool that works with Azure Storage. For more information, see [Azure Storage Client Tools](#).

For logs stored in the App Service file system, the easiest way is to download the ZIP file in the browser at:

- Linux/container apps: <https://<app-name>.scm.azurewebsites.net/api/logs/docker/zip>
- Windows apps: <https://<app-name>.scm.azurewebsites.net/api/dump>

For Linux/container apps, the ZIP file contains console output logs for both the docker host and the docker container. For a scaled-out app, the ZIP file contains one set of logs for each instance. In the App Service file system, these log files are the contents of the `/home/LogFiles` directory.

For Windows apps, the ZIP file contains the contents of the `D:\Home\LogFiles` directory in the App Service file system. It has the following structure:

LOG TYPE	DIRECTORY	DESCRIPTION
Application logs	<i>/LogFiles/Application/</i>	Contains one or more text files. The format of the log messages depends on the logging provider you use.
Failed Request Traces	<i>/LogFiles/W3SVC#####/</i>	Contains XML files, and an XSL file. You can view the formatted XML files in the browser.
Detailed Error Logs	<i>/LogFiles/DetailedErrors/</i>	Contains HTM error files. You can view the HTM files in the browser. Another way to view the failed request traces is to navigate to your app page in the portal. From the left menu, select <b>Diagnose and solve problems</b> , then search for <b>Failed Request Tracing Logs</b> , then click the icon to browse and view the trace you want.
Web Server Logs	<i>/LogFiles/http/RawLogs/</i>	Contains text files formatted using the <a href="#">W3C extended log file format</a> . This information can be read using a text editor or a utility like <a href="#">Log Parser</a> . App Service doesn't support the <code>s-computername</code> , <code>s-ip</code> , or <code>cs-version</code> fields.
Deployment logs	<i>/LogFiles/Git/</i> and <i>/deployments/</i>	Contain logs generated by the internal deployment processes, as well as logs for Git deployments.

## Send logs to Azure Monitor

With the new [Azure Monitor integration](#), you can [create Diagnostic Settings](#) to send logs to Storage Accounts, Event Hubs and Log Analytics.

Subscription \* ⓘ Demo Two Subscription

Resource group ⓘ appsvc-azmon-rg

Diagnostics settings

Name	Storage account	Event hub
No diagnostic settings defined		

[+ Add diagnostic setting](#)

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- AppServiceHTTPLogs
- AppServiceConsoleLogs
- AppServiceAppLogs
- AppServiceFileAuditLogs
- AppServiceAuditLogs
- AllMetrics

**API**

- CORS
- Monitoring**
- Alerts
- Metrics
- Diagnostic settings**
- Logs
- App Service logs
- Log stream
- Process explorer

**Support + troubleshooting**

- Resource health
- App Service Advisor
- New support request

## Supported log types

The following table shows the supported log types and descriptions:

LOG TYPE	WINDOWS	WINDOWS CONTAINER	LINUX	LINUX CONTAINER	DESCRIPTION
AppServiceConsoleLogs	Java SE & Tomcat	Yes	Yes	Yes	Standard output and standard error
AppServiceHTTPLogs	Yes	Yes	Yes	Yes	Web server logs
AppServiceEnvironmentPlatformLogs	Yes	N/A	Yes	Yes	App Service Environment: scaling, configuration changes, and status logs
AppServiceAuditLogs	Yes	Yes	Yes	Yes	Login activity via FTP and Kudu
AppServiceFileAuditLogs	Yes	Yes	TBA	TBA	File changes made to the site content; <b>only available for Premium tier and above</b>

LOG TYPE	WINDOWS	WINDOWS CONTAINER	LINUX	LINUX CONTAINER	DESCRIPTION
AppServiceAppLogs	ASP.NET & Tomcat <sup>1</sup>	ASP.NET & Tomcat <sup>1</sup>	Java SE & Tomcat Blessed Images <sup>2</sup>	Java SE & Tomcat Blessed Images <sup>2</sup>	Application logs
AppServiceIPSecAuditLogs	Yes	Yes	Yes	Yes	Requests from IP Rules
AppServicePlatformLogs	TBA	Yes	Yes	Yes	Container operation logs
AppServiceAntivirusScanAuditLogs	Yes	Yes	Yes	Yes	<a href="#">Anti-virus scan logs</a> using Microsoft Defender; only available for Premium tier

<sup>1</sup> For Tomcat apps, add `TOMCAT_USE_STARTUP_BAT` to the app settings and set it to `false` or `0`. Need to be on the *latest* Tomcat version and use `java.util.logging`.

<sup>2</sup> For Java SE apps, add `WEBSITE_AZMON_PREVIEW_ENABLED` to the app settings and set it to `true` or to `1`.

## Next steps

- [Query logs with Azure Monitor](#)
- [How to Monitor Azure App Service](#)
- [Troubleshooting Azure App Service in Visual Studio](#)
- [Analyze app Logs in HDInsight](#)

# Get resource events in Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

Azure App Service provides built-in tools to monitor the status and health of your resources. Resource events help you understand any changes that were made to your underlying web app resources and take action as necessary. Event examples include: scaling of instances, updates to application settings, restarting of the web app, and many more. In this article, you'll learn how to view [Azure Activity Logs](#) and enable [Event Grid](#) to monitor App Service resource events.

## View Azure Activity Logs

Azure Activity Logs contain resource events emitted by operations taken on the resources in your subscription. Both the user actions in the Azure portal and Azure Resource Manager templates contribute to the events captured by the Activity log.

Azure Activity Logs for App Service details such as:

- What operations were taken on the resources (ex: App Service Plans)
- Who started the operation
- When the operation occurred
- Status of the operation
- Property values to help you research the operation

### What can you do with Azure Activity Logs?

Azure Activity Logs can be queried using the Azure portal, PowerShell, REST API, or CLI. You can send the logs to a storage account, Event Hub, and Log Analytics. You can also analyze them in Power BI or create alerts to stay updated on resource events.

[View and retrieve Azure Activity log events.](#)

## Ship Activity Logs to Event Grid

While Activity logs are user-based, there's a new [Event Grid](#) integration with App Service (preview) that logs both user actions and automated events. With Event Grid, you can configure a handler to react to the said events. For example, use Event Grid to instantly trigger a serverless function to run image analysis each time a new photo is added to a blob storage container.

Alternatively, you can use Event Grid with Logic Apps to process data anywhere, without writing code. Event Grid connects data sources and event handlers.

[View the properties and schema for Azure App Service Events.](#)

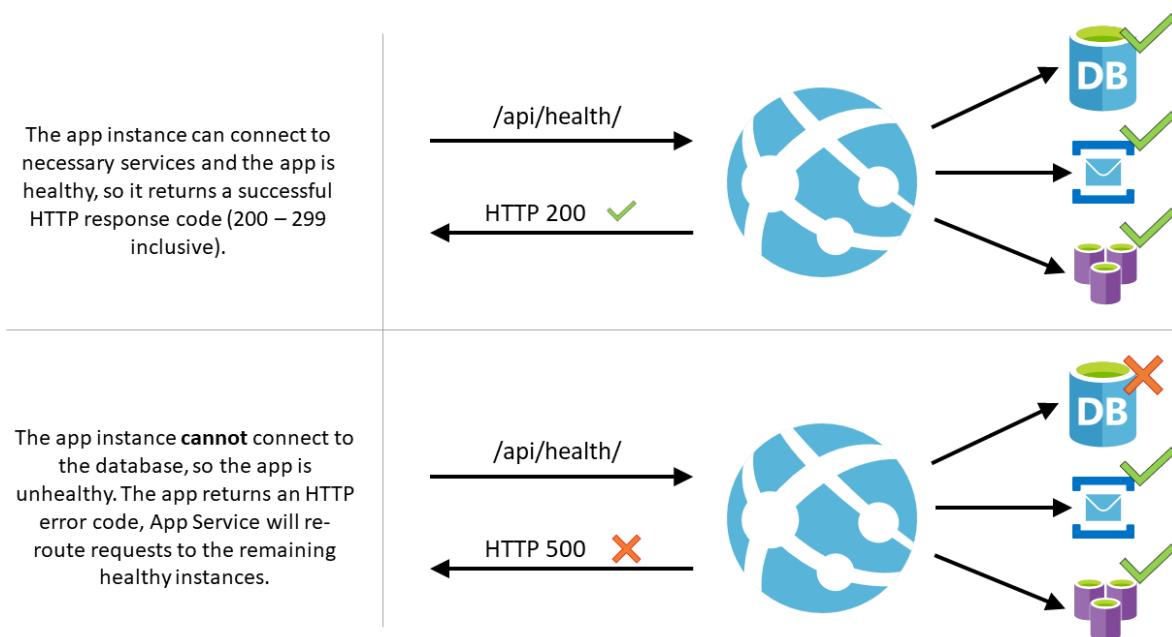
## Next steps

- [Query logs with Azure Monitor](#)
- [How to Monitor Azure App Service](#)
- [Troubleshooting Azure App Service in Visual Studio](#)
- [Analyze app Logs in HDInsight](#)

# Monitor App Service instances using Health check

11/2/2021 • 7 minutes to read • [Edit Online](#)

This article uses Health check in the Azure portal to monitor App Service instances. Health check increases your application's availability by re-routing requests away from unhealthy instances, and replacing instances if they remain unhealthy. Your [App Service plan](#) should be scaled to two or more instances to fully utilize Health check. The Health check path should check critical components of your application. For example, if your application depends on a database and a messaging system, the Health check endpoint should connect to those components. If the application cannot connect to a critical component, then the path should return a 500-level response code to indicate the app is unhealthy.



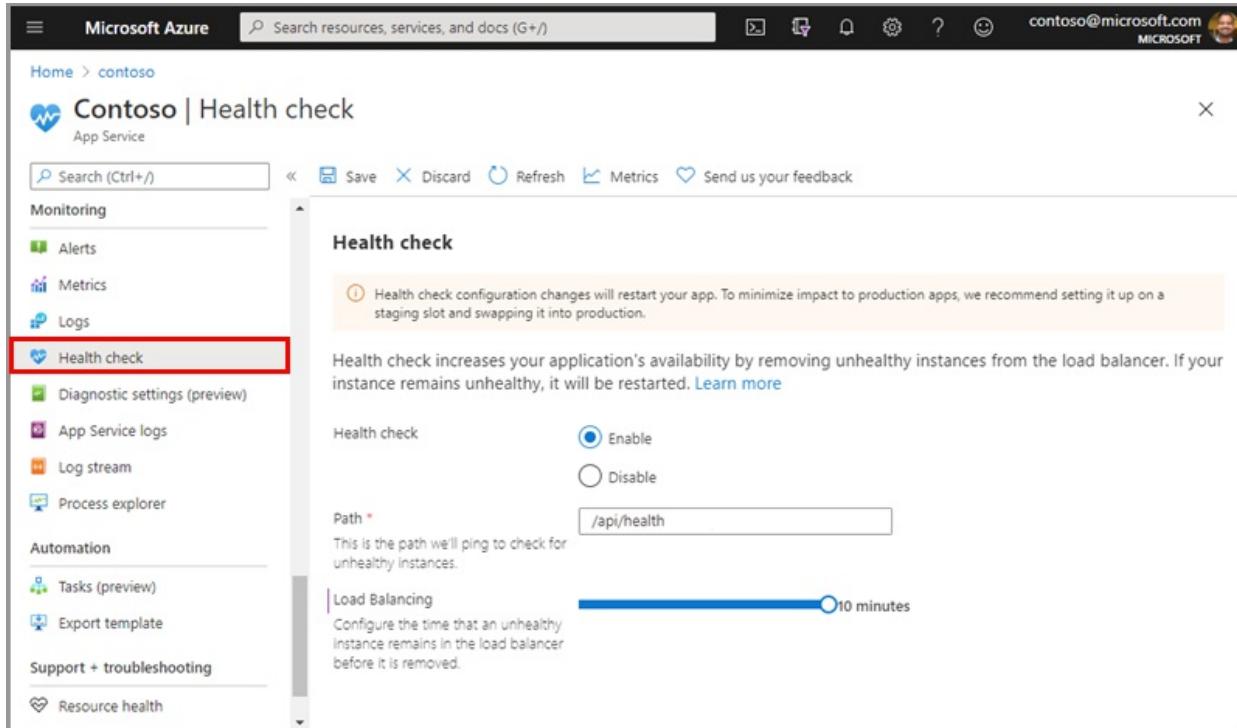
## What App Service does with Health checks

- When given a path on your app, Health check pings this path on all instances of your App Service app at 1-minute intervals.
- If an instance doesn't respond with a status code between 200-299 (inclusive) after two or more requests, or fails to respond to the ping, the system determines it's unhealthy and removes it.
- After removal, Health check continues to ping the unhealthy instance. If the instance begins to respond with a healthy status code (200-299) then the instance is returned to the load balancer.
- If an instance remains unhealthy for one hour, it will be replaced with new instance.
- Furthermore, when scaling up or out, App Service pings the Health check path to ensure new instances are ready.

#### NOTE

- Health check doesn't follow 302 redirects. At most one instance will be replaced per hour, with a maximum of three instances per day per App Service Plan.
- Note, if your health check is giving the status `Waiting for health check response` then the check is likely failing due to an HTTP status code of 307, which can happen if you have HTTPS redirect enabled but have `HTTPS Only` disabled.

## Enable Health Check



The screenshot shows the Azure portal interface for managing an App Service application named 'contoso'. On the left, there's a navigation menu under 'Monitoring' with options like 'Alerts', 'Metrics', 'Logs', and 'Health check' (which is currently selected and highlighted with a red box). The main content area is titled 'Health check' and contains the following information:

- A note: "Health check configuration changes will restart your app. To minimize impact to production apps, we recommend setting it up on a staging slot and swapping it into production."
- A section titled "Health check" with two radio buttons: "Enable" (selected) and "Disable".
- A "Path" input field set to "/api/health".
- A "Load Balancing" section with a slider set to "10 minutes".
- A note: "Health check increases your application's availability by removing unhealthy instances from the load balancer. If your instance remains unhealthy, it will be restarted." followed by a link to "Learn more".

- To enable Health check, browse to the Azure portal and select your App Service app.
- Under **Monitoring**, select **Health check**.
- Select **Enable** and provide a valid URL path on your application, such as `/health` or `/api/health`.
- Click **Save**.

#### Caution

Health check configuration changes restart your app. To minimize impact to production apps, we recommend [configuring staging slots](#) and swapping to production.

## Configuration

In addition to configuring the Health check options, you can also configure the following [app settings](#):

APP SETTING NAME	ALLOWED VALUES	DESCRIPTION
<code>WEBSITE_HEALTHCHECK_MAXPINGFAILURES</code>	2 - 10	The required number of failed requests for an instance to be deemed unhealthy and removed from the load balancer. For example, when set to 2, your instances will be removed after 2 failed pings. (Default value is 10)

APP SETTING NAME	ALLOWED VALUES	DESCRIPTION
WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERS	0 - 100	<p>By default, no more than half of the instances will be excluded from the load balancer at one time to avoid overwhelming the remaining healthy instances. For example, if an App Service Plan is scaled to four instances and three are unhealthy, two will be excluded. The other two instances (one healthy and one unhealthy) will continue to receive requests. In the worst-case scenario where all instances are unhealthy, none will be excluded. To override this behavior, set app setting to a value between 0 and 100. A higher value means more unhealthy instances will be removed (default value is 50).</p>

#### Authentication and security

Health check integrates with App Service's [authentication and authorization features](#). No additional settings are required if these security features are enabled.

If you're using your own authentication system, the Health check path must allow anonymous access. To secure the Health check endpoint, you should first use features such as [IP restrictions](#), [client certificates](#), or a Virtual Network to restrict application access. You can secure the Health check endpoint by requiring the `User-Agent` of the incoming request matches `HealthCheck/1.0`. The User-Agent can't be spoofed since the request would already secured by prior security features.

## Monitoring

After providing your application's Health check path, you can monitor the health of your site using Azure Monitor. From the **Health check** blade in the Portal, click the **Metrics** in the top toolbar. This will open a new blade where you can see the site's historical health status and create a new alert rule. For more information on monitoring your sites, [see the guide on Azure Monitor](#).

## Limitations

- Health check should not be enabled on Premium Functions sites. Due to the rapid scaling of Premium Functions, the Health check requests can cause unnecessary fluctuations in HTTP traffic. Premium Functions have their own internal health probes that are used to inform scaling decisions.
- Health check can be enabled for **Free** and **Shared** App Service Plans so you can have metrics on the site's health and set up alerts, but because **Free** and **Shared** sites cannot scale out, any unhealthy instances will not be replaced. You should scale up to the **Basic** tier or higher so you can scale out to 2 or more instances and utilize the full benefit of Health check. This is recommended for production-facing applications as it will increase your app's availability and performance.

## Frequently Asked Questions

### What happens if my app is running on a single instance?

If your app is only scaled to one instance and becomes unhealthy, it will not be removed from the load balancer because that would take your application down entirely. Scale out to two or more instances to two or more instances to get the re-routing benefit of Health check. If your app is running on a single instance, you can still use Health check's [monitoring](#) feature to keep track of your application's health.

## Why are the Health check request not showing in my frontend logs?

The Health check requests are sent to your site internally, so the request will not show in the [frontend logs](#). This also means the request will have an origin of `127.0.0.1` since it is the request being sent internally. You can add log statements in your Health check code to keep logs of when your Health check path is pinged.

## Are the Health check requests sent over HTTP or HTTPS?

On Windows App Service, the Health check requests will be sent via HTTPS when [HTTPS Only](#) is enabled on the site. Otherwise, they are sent over HTTP. On Linux App Service, the health check requests are only sent over HTTP and cannot be sent over HTTPS at this time.

## What if I have multiple apps on the same App Service Plan?

Unhealthy instances will be always be removed from the load balancer rotation regardless of other apps on the App Service Plan (up to the percentage specified in [WEBSITE\\_HEALTHCHECK\\_MAXUNHEALTHYWORKERPERCENT](#)). When an app on an instance remains unhealthy for over one hour, the instance will only be replaced if all other apps with Health check enabled are also unhealthy. Apps which do not have Health check enabled will not be taken into account.

### Example

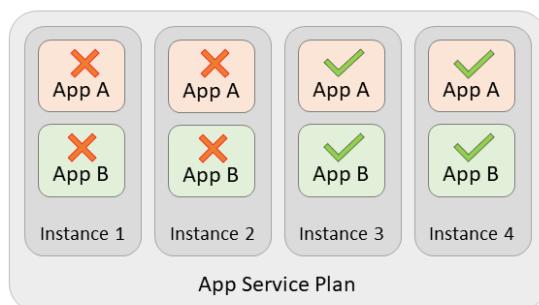
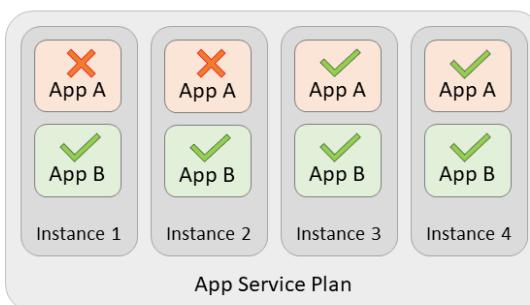
Imagine you have two applications (or one app with a slot) with Health check enabled, called App A and App B. They are on the same App Service Plan and that the Plan is scaled out to 4 instances. If App A becomes unhealthy on two instances, the load balancer will stop sending requests to App A on those two instances. Requests will still be routed to App B on those instances assuming App B is healthy. If App A remains unhealthy for over an hour on those two instances, those instances will only be replaced if App B is also unhealthy on those instances. If App B is healthy, the instance will not be replaced.

App A is unhealthy on instances 1 and 2, so requests will **not** be routed to App A on those instances. App B is healthy on instances 1 and 2, so requests will continue going to App B on those instances.

Instances 1 and 2 will **not** be replaced because App B is healthy on those instances.

Apps A and B are both unhealthy on instances 1 and 2, so requests will **not** be routed to App A or B on those instances.

Instances 1 and 2 will be replaced because all apps on those instances are unhealthy.



### NOTE

If there were another site or slot on the Plan (Site C) without Health check enabled, it would not be taken into consideration for the instance replacement.

## What if all my instances are unhealthy?

In the scenario where all instances of your application are unhealthy, App Service will remove instances from the load balancer up to the percentage specified in [WEBSITE\\_HEALTHCHECK\\_MAXUNHEALTHYWORKERPERCENT](#). In this scenario, taking all unhealthy app instances out of the load balancer rotation would effectively cause an outage for your

application.

### **Does Health Check work on App Service Environments?**

Yes, on App Service Environments (ASEs), the platform will ping your instances on the specified path and remove any unhealthy instances from the load balancer so requests will not be routed to them. However, currently these unhealthy instances will not be replaced with new instances if they remain unhealthy for 1 hour.

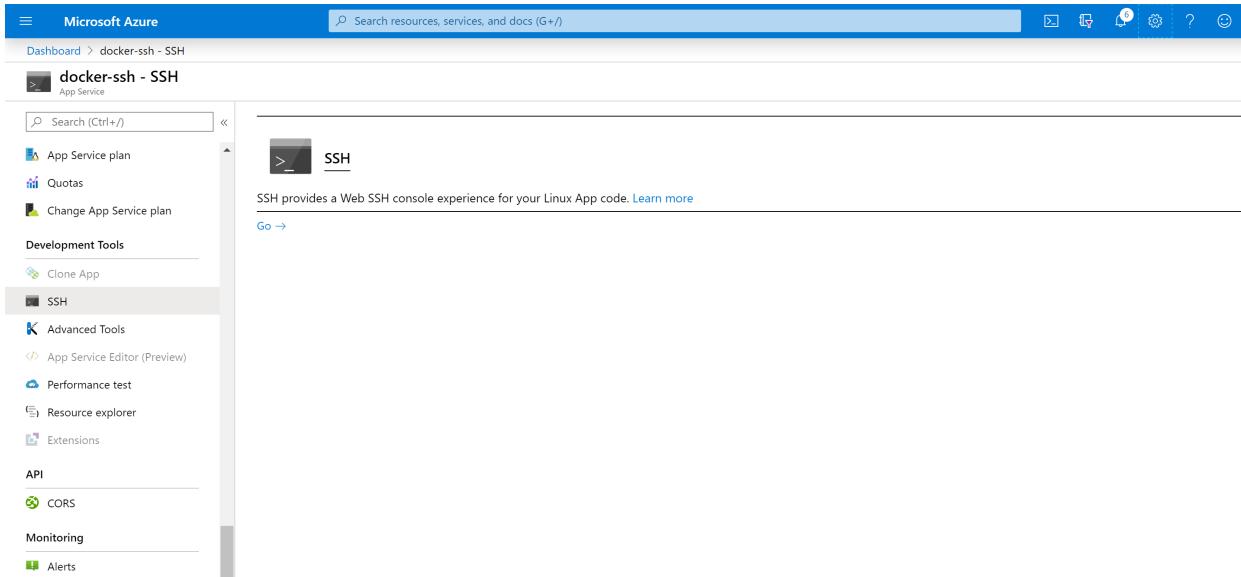
## Next steps

- [Create an Activity Log Alert to monitor all Autoscale engine operations on your subscription](#)
- [Create an Activity Log Alert to monitor all failed Autoscale scale-in/scale-out operations on your subscription](#)
- [Environment variables and app settings reference](#)

# Open an SSH session to a Linux container in Azure App Service

11/2/2021 • 3 minutes to read • [Edit Online](#)

Secure Shell (SSH) is commonly used to execute administrative commands remotely from a command-line terminal. App Service on Linux provides SSH support into the app container.



The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the Microsoft Azure logo and a search bar labeled "Search resources, services, and docs (G+ /)". Below the header, the URL "Dashboard > docker-ssh - SSH" is visible. The main content area is titled "SSH" and contains the text "SSH provides a Web SSH console experience for your Linux App code. Learn more". On the left side, there's a sidebar with several sections: "App Service plan", "Quotas", "Change App Service plan", "Development Tools" (which includes "Clone App", "SSH" which is highlighted in grey, "Advanced Tools", "App Service Editor (Preview)", "Performance test", "Resource explorer", and "Extensions"), "API" (with "CORS"), and "Monitoring" (with "Alerts").

You can also connect to the container directly from your local development machine using SSH and SFTP.

## Open SSH session in browser

To make open a direct SSH session with your container, your app should be running.

Paste the following URL into your browser and replace `<app-name>` with your app name:

```
https://<app-name>.scm.azurewebsites.net/webssh/host
```

If you're not yet authenticated, you're required to authenticate with your Azure subscription to connect. Once authenticated, you see an in-browser shell, where you can run commands inside your container.

```
root@9e933156516f:~# service --status-all
[+] apache2
[-] bootlogs
[-] bootmisc.sh
[-] checkfs.sh
[-] checkroot-bootclean.sh
[-] checkroot.sh
[-] hostname.sh
[?] hwclock.sh
[-] killprocs
[-] motd
[-] mountall-bootclean.sh
[-] mountall.sh
[-] mountdevsubfs.sh
[-] mountkernfs.sh
[-] mountnfs-bootclean.sh
[-] mountnfs.sh
[-] mysql
[-] procps
[-] rc.local
[-] rmmlogin
[-] sendsigs
[+] ssh
[+] udev
[?] udev-finish
[-] umountfs
[-] umountnfs.sh
[-] umountroot
[-] urandom
root@9e933156516f:~#
```

ssh://root@[REDACTED]:2222 SSH CONNECTION ESTABLISHED

## Use SSH support with custom Docker images

See [Configure SSH in a custom container](#).

### Open SSH session from remote shell

#### NOTE

This feature is currently in Preview.

Using TCP tunneling you can create a network connection between your development machine and Web App for Containers over an authenticated WebSocket connection. It enables you to open an SSH session with your container running in App Service from the client of your choice.

To get started, you need to install [Azure CLI](#). To see how it works without installing Azure CLI, open [Azure Cloud Shell](#).

Open a remote connection to your app using the `az webapp create-remote-connection` command. Specify `<subscription-id>`, `<group-name>` and `<app-name>` for your app.

```
az webapp create-remote-connection --subscription <subscription-id> --resource-group <resource-group-name> -n <app-name> &
```

#### TIP

at the end of the command is just for convenience if you are using Cloud Shell. It runs the process in the background so that you can run the next command in the same shell.

#### NOTE

If this command fails, make sure [remote debugging](#) is *disabled* with the following command:

```
az webapp config set --resource-group <resource-group-name> -n <app-name> --remote-debugging-enabled=false
```

The command output gives you the information you need to open an SSH session.

```
Port 21382 is open
SSH is available { username: root, password: Docker! }
Start your favorite client and connect to port 21382
```

Open an SSH session with your container with the client of your choice, using the local port. The following example uses the default [ssh](#) command:

```
ssh root@127.0.0.1 -p <port>
```

When being prompted, type `yes` to continue connecting. You are then prompted for the password. Use `Docker!`, which was shown to you earlier.

```
Warning: Permanently added '[127.0.0.1]:21382' (ECDSA) to the list of known hosts.
root@127.0.0.1's password:
```

Once you're authenticated, you should see the session welcome screen.

```
 _ _ _ \ _ / _ _ _ _ _ _ _ _ _ _ _
 / / \ \ \ / | | \ \ \ \ / \ \ \ \
 / | \ \ / | | / | | \ \ \ \ /
 \ \ \ \ / \ \ \ \ / | | \ \ \ \ >
 \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
 A P P S E R V I C E O N L I N U X
0e690efa93e2:~#
```

You are now connected to your connector.

Try running the [top](#) command. You should be able to see your app's process in the process list. In the example output below, it's the one with `PID 263`.

```
Mem: 1578756K used, 127032K free, 8744K shrd, 201592K buff, 341348K cached
CPU: 3% usr 3% sys 0% nic 92% idle 0% io 0% irq 0% sirq
Load average: 0.07 0.04 0.08 4/765 45738
 PID PPID USER STAT VSZ %VSZ CPU %CPU COMMAND
 1 0 root S 1528 0% 0 0% /sbin/init
 235 1 root S 632m 38% 0 0% PM2 v2.10.3: God Daemon (/root/.pm2)
 263 235 root S 630m 38% 0 0% node /home/site/wwwroot/app.js
 482 291 root S 7368 0% 0 0% sshd: root@pts/0
45513 291 root S 7356 0% 0 0% sshd: root@pts/1
 291 1 root S 7324 0% 0 0% /usr/sbin/sshd
 490 482 root S 1540 0% 0 0% -ash
45539 45513 root S 1540 0% 0 0% -ash
45678 45539 root R 1536 0% 0 0% top
45733 1 root Z 0 0% 0 0% [init]
45734 1 root Z 0 0% 0 0% [init]
45735 1 root Z 0 0% 0 0% [init]
45736 1 root Z 0 0% 0 0% [init]
45737 1 root Z 0 0% 0 0% [init]
45738 1 root Z 0 0% 0 0% [init]
```

## Next steps

You can post questions and concerns on the [Azure forum](#).

For more information on Web App for Containers, see:

- [Introducing remote debugging of Node.js apps on Azure App Service from VS Code](#)
- [Quickstart: Run a custom container on App Service](#)
- [Using Ruby in Azure App Service on Linux](#)
- [Azure App Service Web App for Containers FAQ](#)

# Manage an App Service plan in Azure

11/2/2021 • 3 minutes to read • [Edit Online](#)

An [Azure App Service plan](#) provides the resources that an App Service app needs to run. This guide shows how to manage an App Service plan.

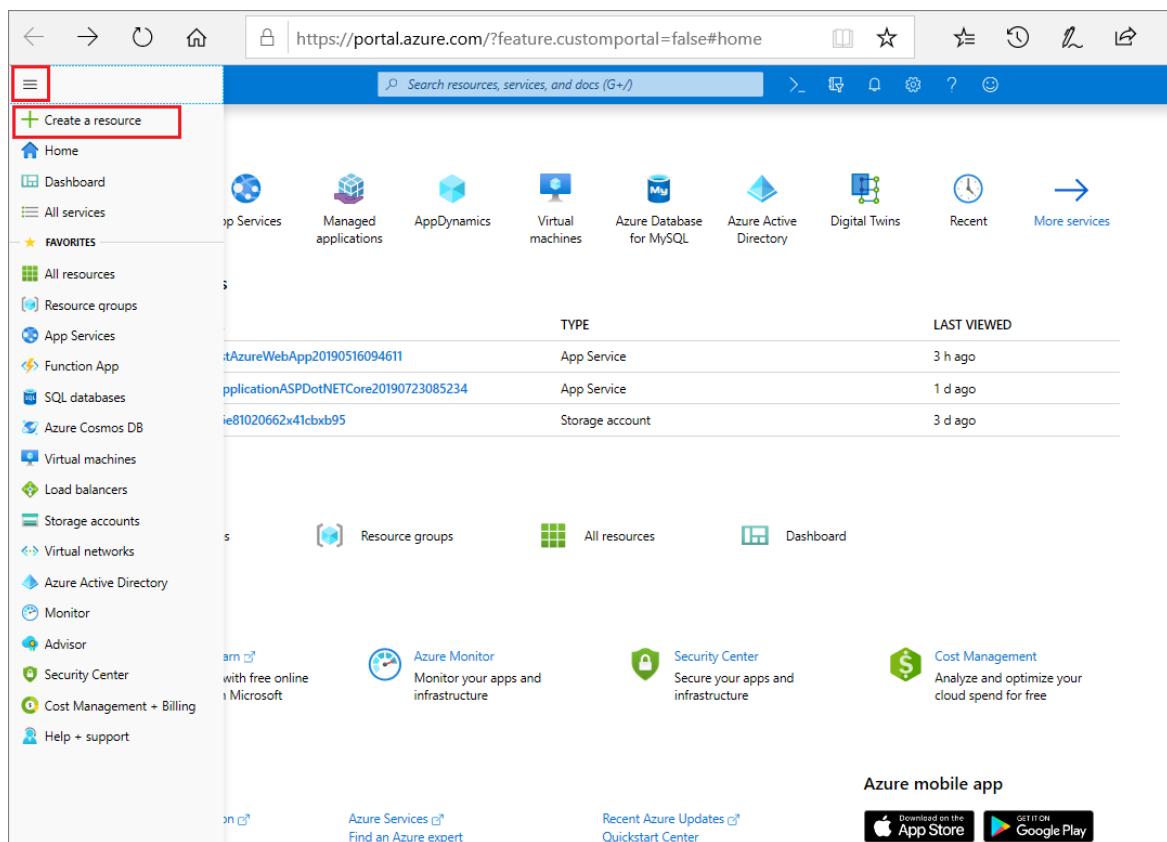
## Create an App Service plan

### TIP

If you want to create a plan in an App Service Environment, you can select it in the **Region** and follow the rest of the steps as described below.

You can create an empty App Service plan, or you can create a plan as part of app creation.

1. In the [Azure portal](#), select **Create a resource**.



The screenshot shows the Azure portal homepage. On the left, there's a navigation sidebar with various service icons like App Services, Managed applications, AppDynamics, Virtual machines, Azure Database for MySQL, Azure Active Directory, Digital Twins, and Recent. Below this is a list of 'FAVORITES' including Home, Dashboard, All services, and All resources. The main content area has a search bar at the top with the URL https://portal.azure.com/?feature.customportal=false#home. Below the search bar is a 'Create a resource' button, which is highlighted with a red box. To the right of the search bar are icons for Home, Dashboard, All resources, and Recent. Below the search bar is a table showing recently viewed resources: 'iAzureWebApp20190516094611' (App Service), 'applicationASPDotNETCore20190723085234' (App Service), and 'ie81020662x41cbxb95' (Storage account). At the bottom of the page are links for Azure Monitor, Security Center, Cost Management, and Azure mobile app, along with download links for the App Store and Google Play.

2. Select **New > Web App** or another kind of App service app.

The screenshot shows the Azure Marketplace interface. At the top, there's a search bar labeled 'Search the Marketplace'. Below it, a navigation bar includes 'Azure Marketplace' and 'See all'. A 'Popular' section lists several services: 'Get started' (selected), 'Recently created', 'AI + Machine Learning', 'Analytics', 'Blockchain', 'Compute', 'Containers', 'Databases', 'Developer Tools', 'DevOps', 'Identity', 'Integration', and 'Internet of Things'. A red box highlights the 'Web App' service, which has a blue icon of a globe and a 'Quickstart tutorial' link.

3. Configure the **Instance Details** section before configuring the App Service plan. Settings such as **Publish** and **Operating Systems** can change the available pricing tiers for your App Service plan. **Region** determines where your App Service plan is created.
4. In the **App Service Plan** section, select an existing plan, or create a plan by selecting **Create new**.

The screenshot shows the 'Web App' creation wizard. The title bar says 'Web App'. The 'Region' dropdown is set to 'Central US', with a note below it: 'Can't find your App Service Plan, try a different region.' The 'App Service Plan' section shows a dropdown menu with '(New) ASP-myResourceGroup-bfdd' selected, and a 'Create new' button highlighted with a red box. Below this, the 'Sku and size' section is set to 'Premium V2 P1v2', with '210 total ACU, 3.5 GB memory' and a 'Change size' link. At the bottom, there are 'Review + create' and 'Next : Tags >' buttons.

5. When creating a plan, you can select the pricing tier of the new plan. In **Sku and size**, select **Change size** to change the pricing tier.

## Move an app to another App Service plan

You can move an app to another App Service plan, as long as the source plan and the target plan are in the *same resource group and geographical region*.

#### NOTE

Azure deploys each new App Service plan into a deployment unit, internally called a webspace. Each region can have many webspaces, but your app can only move between plans that are created in the same webspace. An App Service Environment is an isolated webspace, so apps can be moved between plans in the same App Service Environment, but not between plans in different App Service Environments.

You can't specify the webspace you want when creating a plan, but it's possible to ensure that a plan is created in the same webspace as an existing plan. In brief, all plans created with the same resource group and region combination are deployed into the same webspace. For example, if you created a plan in resource group A and region B, then any plan you subsequently create in resource group A and region B is deployed into the same webspace. Note that plans can't move webspaces after they're created, so you can't move a plan into "the same webspace" as another plan by moving it to another resource group.

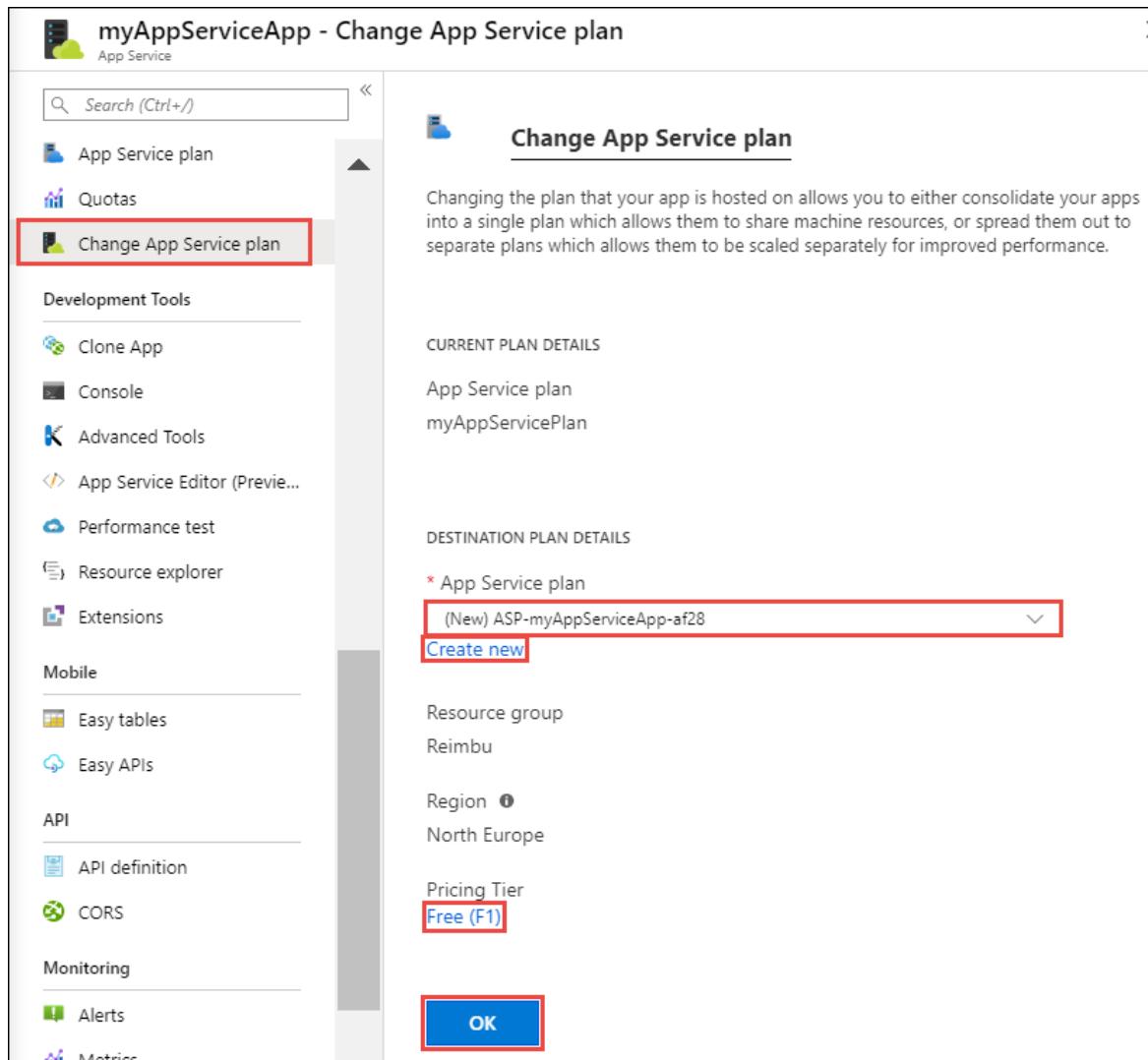
1. In the [Azure portal](#), search for and select **App services** and select the app that you want to move.
2. From the left menu, select **Change App Service plan**.
3. In the **App Service plan** dropdown, select an existing plan to move the app to. The dropdown shows only plans that are in the same resource group and geographical region as the current App Service plan. If no such plan exists, it lets you create a plan by default. You can also create a new plan manually by selecting **Create new**.
4. If you create a plan, you can select the pricing tier of the new plan. In **Pricing Tier**, select the existing tier to change it.

#### IMPORTANT

If you're moving an app from a higher-tiered plan to a lower-tiered plan, such as from **D1** to **F1**, the app may lose certain capabilities in the target plan. For example, if your app uses TLS/SSL certificates, you might see this error message:

```
Cannot update the site with hostname '<app_name>' because its current TLS/SSL configuration 'SNI based SSL enabled' is not allowed in the target compute mode. Allowed TLS/SSL configuration is 'Disabled'.
```

5. When finished, select **OK**.



## Move an app to a different region

The region in which your app runs is the region of the App Service plan it's in. However, you cannot change an App Service plan's region. If you want to run your app in a different region, one alternative is app cloning. Cloning makes a copy of your app in a new or existing App Service plan in any region.

You can find **Clone App** in the **Development Tools** section of the menu.

### IMPORTANT

Cloning has some limitations. You can read about them in [Azure App Service App cloning](#).

## Scale an App Service plan

To scale up an App Service plan's pricing tier, see [Scale up an app in Azure](#).

To scale out an app's instance count, see [Scale instance count manually or automatically](#).

## Delete an App Service plan

To avoid unexpected charges, when you delete the last app in an App Service plan, App Service also deletes the plan by default. If you choose to keep the plan instead, you should change the plan to **Free** tier so you're not charged.

**IMPORTANT**

App Service plans that have no apps associated with them still incur charges because they continue to reserve the configured VM instances.

## Next steps

[Scale up an app in Azure](#)

# Back up your app in Azure

11/2/2021 • 8 minutes to read • [Edit Online](#)

The Backup and Restore feature in [Azure App Service](#) lets you easily create app backups manually or on a schedule. You can configure the backups to be retained up to an indefinite amount of time. You can restore the app to a snapshot of a previous state by overwriting the existing app or restoring to another app.

For information on restoring an app from backup, see [Restore an app in Azure](#).

## What gets backed up

App Service can back up the following information to an Azure storage account and container that you have configured your app to use.

- App configuration
- File content
- Database connected to your app

The following database solutions are supported with backup feature:

- [SQL Database](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)
- [MySQL In-app](#)

### NOTE

Each backup is a complete offline copy of your app, not an incremental update.

## Requirements and restrictions

- The Backup and Restore feature requires the App Service plan to be in the **Standard**, **Premium**, or **Isolated** tier. For more information about scaling your App Service plan to use a higher tier, see [Scale up an app in Azure](#). **Premium** and **Isolated** tiers allow a greater number of daily back ups than **Standard** tier.
- You need an Azure storage account and container in the same subscription as the app that you want to back up. For more information on Azure storage accounts, see [Azure storage account overview](#).
- Backups can be up to 10 GB of app and database content, up to 4GB of which can be the database backup. If the backup size exceeds this limit, you get an error.
- Backups of [TLS enabled Azure Database for MySQL](#) is not supported. If a backup is configured, you will encounter backup failures.
- Backups of [TLS enabled Azure Database for PostgreSQL](#) is not supported. If a backup is configured, you will encounter backup failures.
- In-app MySQL databases are automatically backed up without any configuration. If you make manual settings for in-app MySQL databases, such as adding connection strings, the backups may not work correctly.
- Using a [firewall enabled storage account](#) as the destination for your backups is not supported. If a backup is configured, you will encounter backup failures.
- Using a [private endpoint enabled storage account](#) for backup and restore is not supported.

## Create a manual backup

1. In the [Azure portal](#), navigate to your app's page, select **Backups**. The **Backups** page is displayed.

The screenshot shows the Azure portal interface for an App Service named 'contoso-backup'. The left sidebar has a 'SETTINGS' section with several options: Application settings, Authentication / Authorization, Backups (which is highlighted with a red box), Custom domains, SSL certificates, Networking, and Scale up (App Service plan). A search bar at the top says 'Search (Ctrl+ /)'. The main content area is currently empty.

### NOTE

If you see the following message, click it to upgrade your App Service plan before you can proceed with backups.  
For more information, see [Scale up an app in Azure](#).



2. In the **Backup** page, select **Backup is not configured**. Click [here](#) to configure backup for your app.

The screenshot shows the 'Backup' configuration page. It features a blue cloud icon and the word 'Backup'. Below that is a section titled 'Configure backup to create restorable archive copies of your apps content, configuration and database.' with a 'Learn more' link. A large red box highlights a message box containing a gear icon and the text: 'Backup is not configured. Click here to configure backup for your app.'

3. In the **Backup Configuration** page, click **Storage not configured** to configure a storage account.

The screenshot shows the 'Backup Storage' configuration page. It has a 'Backup Storage' icon and the title 'Backup Storage'. Below it is a section titled 'Select the target container to store your app backup.' A red box highlights a button labeled 'Storage Settings' and 'Storage not configured'.

4. Choose your backup destination by selecting a **Storage Account** and **Container**. The storage account must belong to the same subscription as the app you want to back up. If you wish, you can create a new

storage account or a new container in the respective pages. When you're done, click **Select**.

5. In the **Backup Configuration** page that is still left open, you can configure **Backup Database**, then select the databases you want to include in the backups (SQL Database or MySQL), then click **OK**.

The screenshot shows the 'Backup Database' configuration page. At the top, there's a blue icon with a white 'DB' and a green bar above it. Below the icon, the title 'Backup Database' is displayed. A note below the title says: 'Select the databases to include with your backup. The backup database list is based on the app's configured connection strings. Note: The maximum size of content + database backup cannot exceed 10GB. If your database is large and growing, use Azure Backup for database backup instead.' There are three columns: 'INCLUDE IN BACKUP' (with an unchecked checkbox), 'CONNECTION STRING NAME' (empty), and 'DATABASE TYPE' (empty). A message at the bottom states: 'No supported connection strings of type SQL Database or MySQL found configured in app.'

#### NOTE

For a database to appear in this list, its connection string must exist in the **Connection strings** section of the **Application settings** page for your app.

In-app MySQL databases are automatically backed up without any configuration. If you make settings for in-app MySQL databases manually, such as adding connection strings, the backups may not work correctly.

6. In the **Backup Configuration** page, click **Save**.

7. In the **Backups** page, click **Backup**.

The screenshot shows the 'Backups' page. At the top, there's a blue cloud icon with a white arrow and the title 'Backup'. A note below the title says: 'Configure backup to create restorable archive copies of your app's content, configuration and database. [Learn more](#)'.

A message box contains: 'Backup configured, backup schedule is not configured, configure scheduled backup to automatically take backups.'

Below the message box are two large blue buttons: 'Backup' (with a downward arrow icon) and 'Restore' (with a circular arrow icon).

STATUS	BACkUP TIME	SIZE (MB)
InProgress	Wednesday, October 16, 2019, 6:20:10 PM	0

The 'Backup' button and the first row of the table are highlighted with a red box.

You see a progress message during the backup process.

Once the storage account and container is configured, you can initiate a manual backup at any time. Manual backups are retained indefinitely.

## Configure automated backups

1. In the **Backup Configuration** page, set **Scheduled backup** to **On**.



## Backup Schedule

Configure the schedule for your app backup.

Scheduled backup  On  Off

\* Backup Every  Days

\* Start backup schedule from

\* Retention (Days)

Keep at least one backup  No  Yes

2. Configure the backup schedule as desired and select OK.

## Configure Partial Backups

Sometimes you don't want to back up everything on your app. Here are a few examples:

- You [set up weekly backups](#) of your app that contains static content that never changes, such as old blog posts or images.
- Your app has over 10 GB of content (that's the max amount you can back up at a time).
- You don't want to back up the log files.

Partial backups allow you choose exactly which files you want to back up.

### NOTE

Individual databases in the backup can be 4GB max but the total max size of the backup is 10GB

### Exclude files from your backup

Suppose you have an app that contains log files and static images that have been backup once and are not going to change. In such cases, you can exclude those folders and files from being stored in your future backups. To exclude files and folders from your backups, create a `_backup.filter` file in the `D:\home\site\wwwroot` folder of your app. Specify the list of files and folders you want to exclude in this file.

You can access your files by navigating to <https://<app-name>.scm.azurewebsites.net/DebugConsole>. If prompted, sign in to your Azure account.

Identify the folders that you want to exclude from your backups. For example, you want to filter out the highlighted folder and files.

## ... / Images + | 6 items

	Name
	2013
	2014
	2015
	Products
	bkg.png
	brand.png

Create a file called `_backup.filter` and put the preceding list in the file, but remove `D:\home`. List one directory or file per line. So the content of the file should be:

```
\site\wwwroot\Images\brand.png
\site\wwwroot\Images\2014
\site\wwwroot\Images\2013
```

Upload `_backup.filter` file to the `D:\home\site\wwwroot\` directory of your site using [ftp](#) or any other method. If you wish, you can create the file directly using Kudu [DebugConsole](#) and insert the content there.

Run backups the same way you would normally do it, [manually](#) or [automatically](#). Now, any files and folders that are specified in `_backup.filter` is excluded from the future backups scheduled or manually initiated.

### NOTE

You restore partial backups of your site the same way you would [restore a regular backup](#). The restore process does the right thing.

When a full backup is restored, all content on the site is replaced with whatever is in the backup. If a file is on the site, but not in the backup it gets deleted. But when a partial backup is restored, any content that is located in one of the restricted directories, or any restricted file, is left as is.

## How backups are stored

After you have made one or more backups for your app, the backups are visible on the [Containers](#) page of your storage account, and your app. In the storage account, each backup consists of a `.zip` file that contains the backup data and an `.xml` file that contains a manifest of the `.zip` file contents. You can unzip and browse these files if you want to access your backups without actually performing an app restore.

The database backup for the app is stored in the root of the `.zip` file. For SQL Database, this is a BACPAC file (no file extension) and can be imported. To create a database in Azure SQL Database based on the BACPAC export, see [Import a BACPAC file to create a database in Azure SQL Database](#).

### WARNING

Altering any of the files in your `websitebackups` container can cause the backup to become invalid and therefore non-restorable.

## Troubleshooting

The **Backups** page shows you the status of each backup. If you click on a failed backup, you can get log details regarding the failure. Use the following table to help you troubleshoot your backup. If the failure isn't documented in the table, open a support ticket.

ERROR	FIX
Storage access failed.	Delete backup schedule and reconfigure it. Or, reconfigure the backup storage.
The website + database size exceeds the {0} GB limit for backups. Your content size is {1} GB.	<b>Exclude some files</b> from the backup, or remove the database portion of the backup and use externally offered backups instead.
Error occurred while connecting to the database {0} on server {1}: Authentication to host '{1}' for user '<username>' using method 'mysql_native_password' failed with message: Unknown database '<db-name>'	Update database connection string.
Cannot resolve {0}. {1} (CannotResolveStorageAccount)	Delete the backup schedule and reconfigure it.
Login failed for user '{0}'.	Update the database connection string.
Create Database copy of {0} ({1}) threw an exception. Could not create Database copy.	Use an administrative user in the connection string.
The server principal "<name>" is not able to access the database "master" under the current security context. Cannot open database "master" requested by the login. The login failed. Login failed for user '<name>'.	Use an administrative user in the connection string.
A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server).	Check that the connection string is valid. Allow the app's <b>outbound IPs</b> in the database server settings.
Cannot open server "<name>" requested by the login. The login failed.	Check that the connection string is valid.
Missing mandatory parameters for valid Shared Access Signature.	Delete the backup schedule and reconfigure it.
SSL connection is required. Please specify SSL options and retry. when trying to connect.	Use the built-in backup feature in Azure MySQL or Azure Postgresql instead.

## Automate with scripts

You can automate backup management with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

For samples, see:

- [Azure CLI samples](#)
- [Azure PowerShell samples](#)

## Next Steps

For information on restoring an app from a backup, see [Restore an app in Azure](#).

# Restore an app in Azure

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to restore an app in [Azure App Service](#) that you have previously backed up (see [Back up your app in Azure](#)). You can restore your app with its linked databases on-demand to a previous state, or create a new app based on one of your original app's backups. Azure App Service supports the following databases for backup and restore:

- [SQL Database](#)
- [Azure Database for MySQL](#)
- [Azure Database for PostgreSQL](#)
- [MySQL in-app](#)

Restoring from backups is available to apps running in **Standard** and **Premium** tier. For information about scaling up your app, see [Scale up an app in Azure](#). **Premium** tier allows a greater number of daily backups to be performed than **Standard** tier.

## Restore an app from an existing backup

1. On the **Settings** page of your app in the Azure portal, click **Backups** to display the **Backups** page. Then click **Restore**.

The screenshot shows the 'Backups' page for an app named 'contoso-backup'. The left sidebar has sections for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart, Deployment credentials, Deployment slots, Deployment options, Continuous Delivery (Preview)), and Settings (Application settings, Authentication / Authorization, Backups). The 'Backups' link in the Settings section is highlighted with a red box. The main content area has a 'Backup' section with a warning message about backup configuration. Below it are 'Configure', 'Backup', and 'Restore' buttons, where 'Restore' is also highlighted with a red box. A table lists a single backup entry: a green checkmark icon, 'S...', 'Thursday, June 8, 2017..', '0.06 MB', and a 'Restore' link.

STATUS	BACKUP TIME	SIZE (MB)
✓ S...	Thursday, June 8, 2017..	0.06

2. In the **Restore** page, first select the backup source.

Select from either a Backup on the app, zip file of a valid backup from a storage container or select a snapshot of the app.

Restore source ⓘ App backup Storage

Select the Backup to Restore ⓘ

Backed up Thursday, June 8, 2017, 5:22:04 PM PDT

The **App backup** option shows you all the existing backups of the current app, and you can easily select one. The **Storage** option lets you select any backup ZIP file from any existing Azure Storage account and container in your subscription. If you're trying to restore a backup of another app, use the **Storage** option.

3. Then, specify the destination for the app restore in **Restore destination**.

Select to override the current app or an existing app to restore content.

Restore destination ⓘ Overwrite New or existing app

**WARNING**  
If you choose **Overwrite**, all existing data in your current app is erased and overwritten. Before you click **OK**, make sure that it is exactly what you want to do.

**WARNING**  
If the App Service is writing data to the database while you are restoring it, it may result in symptoms such as violation of PRIMARY KEY and data loss. It is suggested to stop the App Service first before you start to restore the database.

You can select **Existing App** to restore the app backup to another app in the same resource group. Before you use this option, you should have already created another app in your resource group with mirroring database configuration to the one defined in the app backup. You can also Create a **New** app to restore your content to.

4. Click **OK**.

## Download or delete a backup from a storage account

1. From the main **Browse** page of the Azure portal, select **Storage accounts**. A list of your existing storage accounts is displayed.
2. Select the storage account that contains the backup that you want to download or delete. The page for the storage account is displayed.
3. In the storage account page, select the container you want

The screenshot shows the Azure Storage Account - Blob blade for the storage account 'cephalinstorage4'. At the top, there are navigation icons for Settings, Delete, Container, and Refresh. Below that is the 'Essentials' section with the following details:

Resource group	Performance/Access tier
<a href="#">cephalin-appwithsql</a>	Standard/Hot
Status	Replication
Primary: Available	Locally-redundant storage (LRS)
Location	Blob service endpoint
West Europe	<a href="https://cephalinstorage4.blob.core.windows.net">https://cephalinstorage4.blob.core.windows..</a>
Subscription name	
<a href="#">Visual Studio Ultimate with MSDN</a>	<a href="#">Edit</a>
Subscription ID	

Below the essentials section is a search bar labeled 'Search containers by prefix' and a table listing containers:

NAME	URL	LAST MODIFIED	...
backups	<a href="https://cephalinstorage4.blob.core.windows.net/backups">https://cephalinstorage4.blob.core.windows.net/backups</a>	7/6/2016, 2:00:16 PM	<a href="#">...</a>

At the bottom right of the essentials section is a link 'All settings →'.

4. Select backup file you want to download or delete.

The screenshot shows the Azure Storage Explorer interface with the 'backups' container selected. At the top, there are four action buttons: Refresh, Delete container, Properties, and Access policy. Below the buttons is a search bar with the placeholder text 'Search blobs by prefix (case-sensitive)'. The main area displays a table of blobs with the following columns: NAME, MODIFIED, BLOB TYPE, and SIZE. There are three blobs listed:

NAME	MODIFIED	BLOB TYPE	SIZE
cephalin-appwithsql_201607061211.log	7/6/2016, 2:15:58...	Block blob	272 B
cephalin-appwithsql_201607061211.xml	7/6/2016, 2:15:58...	Block blob	793 B
cephalin-appwithsql_201607061211.zip	7/6/2016, 2:15:58...	Block blob	151.11 KB

5. Click **Download** or **Delete** depending on what you want to do.

## Monitor a restore operation

To see details about the success or failure of the app restore operation, navigate to the **Activity Log** page in the Azure portal.

Scroll down to find the desired restore operation and click to select it.

The details page displays the available information related to the restore operation.

## Automate with scripts

You can automate backup management with scripts, using the [Azure CLI](#) or [Azure PowerShell](#).

For samples, see:

- [Azure CLI samples](#)
- [Azure PowerShell samples](#)

# Restore an app in Azure from a snapshot

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to restore an app in [Azure App Service](#) from a snapshot. You can restore your app to a previous state, based on one of your app's snapshots. You do not need to enable snapshots, the platform automatically saves a snapshot of all apps for data recovery purposes.

Snapshots are incremental shadow copies of your App Service app. When your app is in Premium tier or higher, App Service takes periodic snapshots of both the app's content and its configuration. They offer several advantages over [standard backups](#):

- No file copy errors due to file locks.
- Higher maximum snapshot size (30GB).
- No configuration required for supported pricing tiers.
- Snapshots can be restored to a new App Service app in any Azure region.

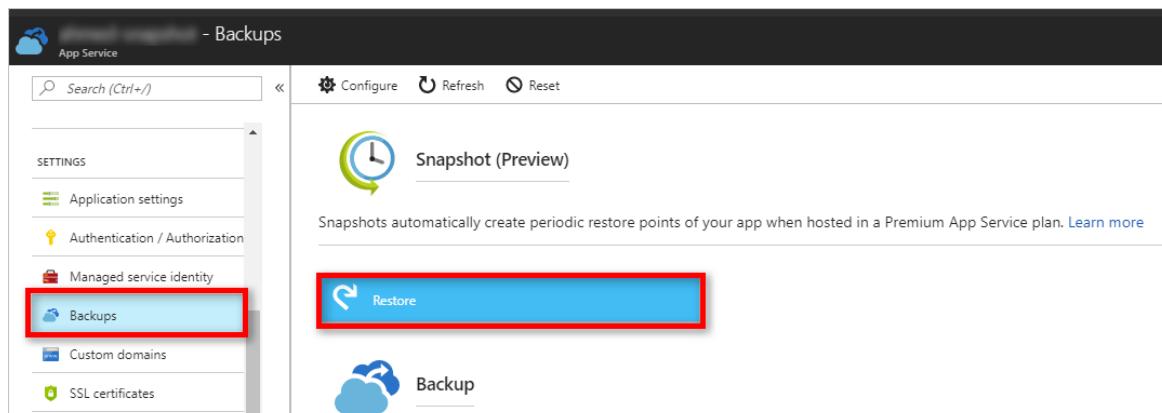
Restoring from snapshots is available to apps running in Premium tier or higher. For information about scaling up your app, see [Scale up an app in Azure](#).

## Limitations

- Maximum supported size for snapshot restore is 30GB. Snapshot restore fails if your storage size is greater than 30GB. To reduce your storage size, consider moving files like logs, images, audios, and videos to [Azure Storage](#), for example.
- Any connected database that [standard backup](#) supports or [mounted Azure storage](#) is *not* included in the snapshot. Consider using the native backup capabilities of the connected Azure service (for example, [SQL Database](#) and [Azure Files](#)).
- App Service stops the target app or target slot while restoring a snapshot. To minimize downtime for the production app, restore the snapshot to a [staging slot](#) first, then swap into production.
- Snapshots for the last 30 days are available. The retention period and snapshot frequency are not configurable.
- App Services running on an App Service environment do not support snapshots.

## Restore an app from a snapshot

1. On the **Settings** page of your app in the [Azure portal](#), click **Backups** to display the **Backups** page. Then click **Restore** under the **Snapshot(Preview)** section.



2. In the **Restore** page, select the snapshot to restore.

Select from either a Backup on the app or zip file of a valid backup from a storage container.

Restore source ⓘ App backup Storage Snapshot (Preview)

Snapshot (Preview) ⓘ

Snapshot taken at Monday, October 23, 2017, 10:41:49 PM PDT

3. Specify the destination for the app restore in **Restore destination**.

Select to overwrite the current app or an existing app to restore content.

Restore destination ⓘ Overwrite New or existing app

#### WARNING

As a best practice we recommend restoring to a new slot then performing a swap. If you choose **Overwrite**, all existing data in your app's current file system is erased and overwritten. Before you click **OK**, make sure that it is what you want to do.

#### NOTE

Due to current technical limitations, you can only restore to apps in the same scale unit. This limitation will be removed in a future release.

You can select **Existing App** to restore to a slot. Before you use this option, you should have already created a slot in your app.

4. You can choose to restore your site configuration.

Advanced settings for restoring an app backup with options.

Restore site configuration ⓘ No Yes

5. Click **OK**.

# Azure App Service App Cloning Using PowerShell

11/2/2021 • 5 minutes to read • [Edit Online](#)

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

With the release of Microsoft Azure PowerShell version 1.1.0, a new option has been added to `New-AzWebApp` that lets you clone an existing App Service app to a newly created app in a different region or in the same region. This option enables customers to deploy a number of apps across different regions quickly and easily.

App cloning is supported for Standard, Premium, Premium V2, and Isolated app service plans. The new feature uses the same limitations as App Service Backup feature, see [Back up an app in Azure App Service](#).

## Cloning an existing app

Scenario: An existing app in South Central US region, and you want to clone the contents to a new app in North Central US region. It can be accomplished by using the Azure Resource Manager version of the PowerShell cmdlet to create a new app with the `-SourceWebApp` option.

Knowing the resource group name that contains the source app, you can use the following PowerShell command to get the source app's information (in this case named `source-webapp`):

```
$srcapp = Get-AzWebApp -ResourceGroupName SourceAzureResourceGroup -Name source-webapp
```

To create a new App Service Plan, you can use `New-AzAppServicePlan` command as in the following example

```
New-AzAppServicePlan -Location "North Central US" -ResourceGroupName DestinationAzureResourceGroup -Name DestinationAppServicePlan -Tier Standard
```

Using the `New-AzWebApp` command, you can create the new app in the North Central US region, and tie it to an existing App Service Plan. Moreover, you can use the same resource group as the source app, or define a new resource group, as shown in the following command:

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp
```

To clone an existing app including all associated deployment slots, you need to use the `IncludeSourceWebAppSlots` parameter. Note that the `IncludeSourceWebAppSlots` parameter is only supported for cloning an entire app including all of its slots. The following PowerShell command demonstrates the use of that parameter with the `New-AzWebApp` command:

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -IncludeSourceWebAppSlots
```

To clone an existing app within the same region, you need to create a new resource group and a new app service

plan in the same region, and then use the following PowerShell command to clone the app:

```
$destapp = New-AzWebApp -ResourceGroupName NewAzureResourceGroup -Name dest-webapp -Location "South Central US" -AppServicePlan NewAppServicePlan -SourceWebApp $srcapp
```

## Cloning an existing App to an App Service Environment

Scenario: An existing app in South Central US region, and you want to clone the contents to a new app to an existing App Service Environment (ASE).

Knowing the resource group name that contains the source app, you can use the following PowerShell command to get the source app's information (in this case named `source-webapp`):

```
$srcapp = Get-AzWebApp -ResourceGroupName SourceAzureResourceGroup -Name source-webapp
```

Knowing the ASE's name, and the resource group name that the ASE belongs to, you can create the new app in the existing ASE, as shown in the following command:

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "North Central US" -AppServicePlan DestinationAppServicePlan -ASEName DestinationASE -ASEResourceGroupName DestinationASEResourceGroupName -SourceWebApp $srcapp
```

The `Location` parameter is required due to legacy reason, but it is ignored when you create the app in an ASE.

## Cloning an existing App Slot

Scenario: You want to clone an existing deployment slot of an app to either a new app or a new slot. The new app can be in the same region as the original app slot or in a different region.

Knowing the resource group name that contains the source app, you can use the following PowerShell command to get the source app slot's information (in this case named `source-appslot`) tied to `source-app`:

```
$srcappslot = Get-AzWebAppSlot -ResourceGroupName SourceAzureResourceGroup -Name source-app -Slot source-appslot
```

The following command demonstrates creating a clone of the source app to a new app:

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-app -Location "North Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcappslot
```

## Configuring Traffic Manager while cloning an app

Creating multi-region apps and configuring Azure Traffic Manager to route traffic to all these apps, is an important scenario to ensure that customers' apps are highly available. When cloning an existing app, you have the option to connect both apps to either a new traffic manager profile or an existing one. Only Azure Resource Manager version of Traffic Manager is supported.

### **Creating a new Traffic Manager profile while cloning an app**

Scenario: You want to clone an app to another region, while configuring an Azure Resource Manager traffic manager profile that includes both apps. The following command demonstrates creating a clone of the source app to a new app while configuring a new Traffic Manager profile:

```
$destapp = New-AzWebApp -ResourceGroupName DestinationAzureResourceGroup -Name dest-webapp -Location "South Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -TrafficManagerProfileName newTrafficManagerProfile
```

## Adding new cloned app to an existing Traffic Manager profile

Scenario: You already have an Azure Resource Manager traffic manager profile and want to add both apps as endpoints. To do so, you first need to assemble the existing traffic manager profile ID. You need the subscription ID, the resource group name, and the existing traffic manager profile name.

```
$TMProfileID = "/subscriptions/<Your subscription ID goes here>/resourceGroups/<Your resource group name goes here>/providers/Microsoft.TrafficManagerProfiles/ExistingTrafficManagerProfileName"
```

After having the traffic manger ID, the following command demonstrates creating a clone of the source app to a new app while adding them to an existing Traffic Manager profile:

```
$destapp = New-AzWebApp -ResourceGroupName <Resource group name> -Name dest-webapp -Location "South Central US" -AppServicePlan DestinationAppServicePlan -SourceWebApp $srcapp -TrafficManagerProfileId $TMProfileID
```

### NOTE

If you are receiving an error that states, "SSL validation on the traffic manager hostname is failing" then we suggest you use -IgnoreCustomHostNames attribute in the powershell cmdlet while performing the clone operation or else use the portal.

## Current Restrictions

Here are the known restrictions of app cloning:

- Auto scale settings are not cloned
- Backup schedule settings are not cloned
- VNET settings are not cloned
- App Insights are not automatically set up on the destination app
- Easy Auth settings are not cloned
- Kudu Extension are not cloned
- TiP rules are not cloned
- Database content is not cloned
- Outbound IP Addresses changes if cloning to a different scale unit
- Not available for Linux Apps
- Managed Identities are not cloned

## References

- [App Service Cloning](#)
- [Back up an app in Azure App Service](#)
- [Azure Resource Manager support for Azure Traffic Manager Preview](#)
- [Introduction to App Service Environment](#)
- [Using Azure PowerShell with Azure Resource Manager](#)

# Restore deleted App Service app Using PowerShell

11/2/2021 • 2 minutes to read • [Edit Online](#)

If you happened to accidentally delete your app in Azure App Service, you can restore it using the commands from the [Az PowerShell module](#).

## NOTE

- Deleted apps are purged from the system 30 days after the initial deletion. After an app is purged, it can't be recovered.
- Undelete functionality isn't supported for the Consumption plan.
- App Service apps running in an App Service Environment don't support snapshots. Therefore, undelete functionality and clone functionality aren't supported for App Service apps running in an App Service Environment.

## Re-register App Service resource provider

Some customers might come across an issue where retrieving the list of deleted apps fails. To resolve the issue, run the following command:

```
Register-AzResourceProvider -ProviderNamespace "Microsoft.Web"
```

## List deleted apps

To get the collection of deleted apps, you can use `Get-AzDeletedWebApp`.

For details on a specific deleted app you can use:

```
Get-AzDeletedWebApp -Name <your_deleted_app> -Location <your_deleted_app_location>
```

The detailed information includes:

- DeletedSiteId**: Unique identifier for the app, used for scenarios where multiple apps with the same name have been deleted
- SubscriptionID**: Subscription containing the deleted resource
- Location**: Location of the original app
- ResourceGroupName**: Name of the original resource group
- Name**: Name of the original app.
- Slot**: the name of the slot.
- Deletion Time**: When was the app deleted

## Restore deleted app

## NOTE

`Restore-AzDeletedWebApp` isn't supported for function apps.

Once the app you want to restore has been identified, you can restore it using `Restore-AzDeletedWebApp`.

```
Restore-AzDeletedWebApp -TargetResourceGroupName <my_rg> -Name <my_app> -TargetAppServicePlanName <my_asp>
```

#### NOTE

Deployment slots are not restored as part of your app. If you need to restore a staging slot, use the `-Slot <slot-name>` flag.

The inputs for command are:

- **Target Resource Group:** Target resource group where the app will be restored
- **Name:** Name for the app, should be globally unique.
- **TargetAppServicePlanName:** App Service plan linked to the app

By default `Restore-AzDeletedWebApp` will restore both your app configuration as well any content. If you want to only restore content, you use the `-RestoreContentOnly` flag with this commandlet.

#### NOTE

If the app was hosted on and then deleted from an App Service Environment, it can be restored only if the corresponding App Service Environment still exists.

You can find the full commandlet reference here: [Restore-AzDeletedWebApp](#).

# Move an App Service app to another region

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article describes how to bring App Service resources back online in a different Azure region during a disaster that impacts an entire Azure region. When a disaster brings an entire Azure region offline, all App Service apps hosted in that region are placed in disaster recovery mode. Features are available to help you restore the app to a different region or recover files from the impacted app.

App Service resources are region-specific and can't be moved across regions. You must restore the app to a new app in a different region, and then create mirroring configurations or resources for the new app.

## Prerequisites

- None. [Restoring from snapshot](#) usually requires **Premium** tier, but in disaster recovery mode, it's automatically enabled for your impacted app, regardless which tier the impacted app is in.

## Prepare

Identify all the App Service resources that the impacted app currently uses. For example:

- App Service apps
- [App Service plans](#)
- [Deployment slots](#)
- [Custom domains purchased in Azure](#)
- [TLS/SSL certificates](#)
- [Azure Virtual Network integration](#)
- [Hybrid connections.](#)
- [Managed identities](#)
- [Backup settings](#)

Certain resources, such as imported certificates or hybrid connections, contain integration with other Azure services. For information on how to move those resources across regions, see the documentation for the respective services.

## Restore app to a different region

1. Create a new App Service app in a *different* Azure region than the impacted app. This is the target app in the disaster recovery scenario.
2. In the [Azure portal](#), navigate to the impacted app's management page. In a failed Azure region, the impacted app shows a warning text. Click the warning text.

Home >

Resource group ([change](#))  
DRTesting

Status  
---

Location  
eastasiastage

Subscription ([change](#))  
DR OGF

Subscription ID  
00000000-0000-0000-0000-000000000000

3. In the **Restore Backup** page, configure the restore operation according to the following table. When finished, click **OK**.

SETTING	VALUE	DESCRIPTION
Snapshot (Preview)	Select a snapshot.	The two most recent snapshots are available.
Restore destination	Existing app	Click the note below that says <b>Click here to change the restore destination app</b> and select the target app. In a disaster scenario, you can only restore the snapshot to an app in a different Azure region.
Restore site configuration	Yes	

## Restore Backup

X



### Select Backup to Restore

Select a Backup on the app.

#### Snapshot (Preview) ⓘ

Snapshot taken at Tuesday, June 2, 2020, 11:41:15 AM PDT



### Select a target App Service App

Select to overwrite the current app or an existing app to restore content. Snapshots can only be restored to apps in the same App Service Plan.



Overwriting the source app will result in data loss and will also cause extended period of downtime while the app is being restored. Make sure you have a backup of the current app content before overwriting the current app.

#### Restore destination ⓘ

Overwrite Existing app



No restore destination app selected. Click here to select the restore destination app.

Select a slot to restore the backup to



### Advanced Settings

Advanced settings for restoring an app backup with options.

#### Restore site configuration

No Yes

OK

4. Configure [everything else](#) in the target app to mirror the impacted app and verify your configuration.

5. When you're ready for the custom domain to point to the target app, [remap the domain name](#).

## Recover app content only

If you only want to recover the files from the impacted app without restoring it, use the following steps:

1. In the [Azure portal](#), navigate to the impacted app's management page and click [Get publish profile](#).

Home >

drtestsite ⚡

App Service

» Browse Stop Swap Restart Delete [Get publish profile](#) ...

⚠ We are unable to retrieve comprehensive information about your app because it is operating in a limited state and may be down. Click here to restore your app contents from a snapshot.

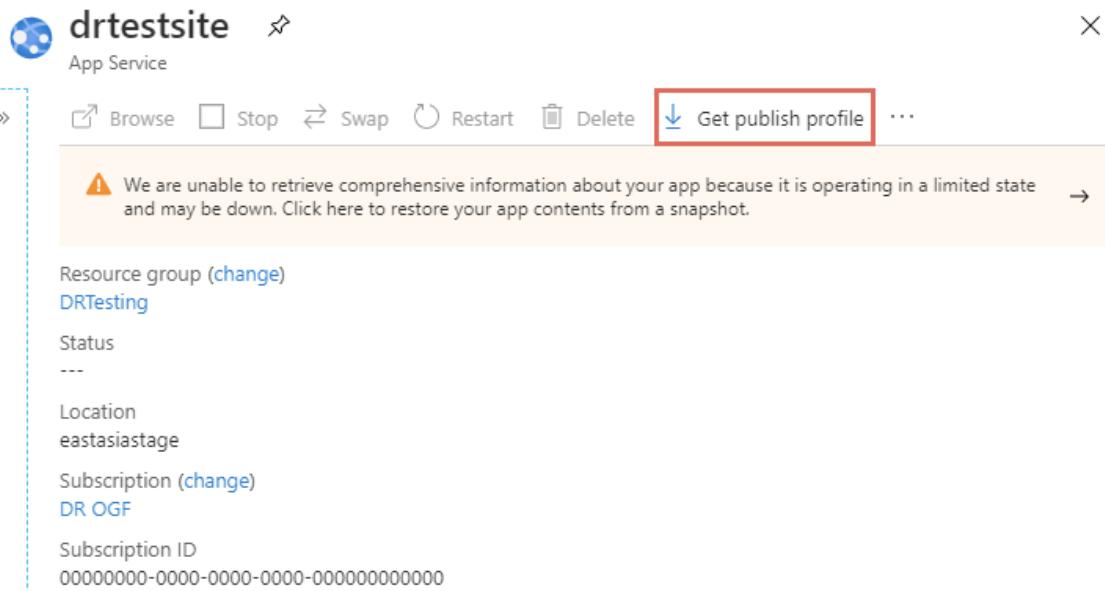
Resource group ([change](#))  
DRTesting

Status  
---

Location  
eastasiastage

Subscription ([change](#))  
DR OGF

Subscription ID  
00000000-0000-0000-0000-000000000000



2. Open the downloaded file and find the publishing profile that contains `ReadOnly - FTP` in its name. This is the disaster recovery profile. For example:

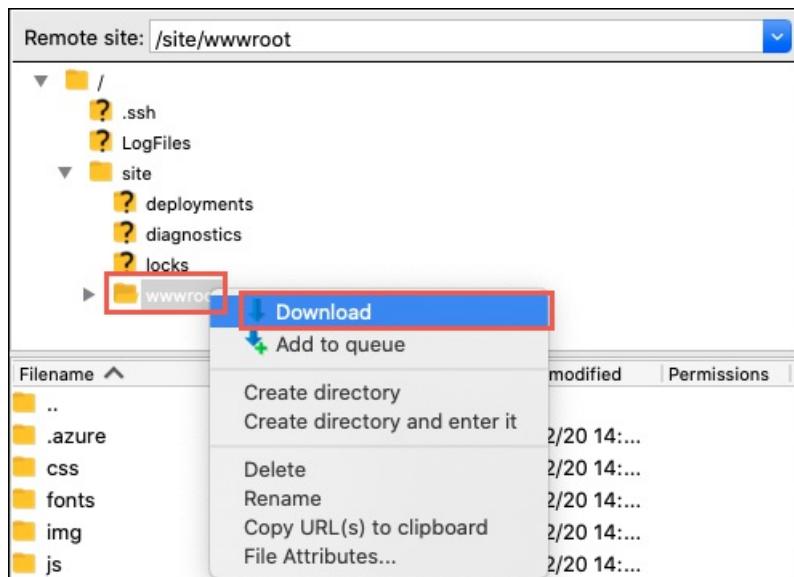
```
<publishProfile profileName="%app-name% - ReadOnly - FTP" publishMethod="FTP" publishUrl="ftp://%ftp-site%/site/wwwroot" ftpPassiveMode="True" userName="%app-name%\%app-name%" userPWD="" destinationAppUrl="http://%app-name%.azurewebsites.net" SQLServerDBConnectionString="" mySQLDBConnectionString="" hostingProviderForumLink="" controlPanelLink="http://windows.azure.com" webSystem="WebSites">
 <databases />
</publishProfile>
```

Copy three attribute values:

- `publishUrl` : the FTP hostname
- `userName` and `userPWD` : the FTP credentials

3. Use the FTP client of your choice, connect to the impacted app's FTP host using the hostname and credentials.

4. Once connected, download the entire `/site/wwwroot` folder. The following screenshot shows how you download in [FileZilla](#).



## Next steps

[Restore an app in Azure from a snapshot](#)

# Set up an Azure Arc-enabled Kubernetes cluster to run App Service, Functions, and Logic Apps (Preview)

11/2/2021 • 8 minutes to read • [Edit Online](#)

If you have an [Azure Arc-enabled Kubernetes cluster](#), you can use it to create an [App Service enabled custom location](#) and deploy web apps, function apps, and logic apps to it.

Azure Arc-enabled Kubernetes lets you make your on-premises or cloud Kubernetes cluster visible to App Service, Functions, and Logic Apps in Azure. You can create an app and deploy to it just like another Azure region.

## Prerequisites

If you don't have an Azure account, [sign up today](#) for a free account.

## Add Azure CLI extensions

Launch the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

Because these CLI commands are not yet part of the core CLI set, add them with the following commands.

```
az extension add --upgrade --yes --name connectedk8s
az extension add --upgrade --yes --name k8s-extension
az extension add --upgrade --yes --name customlocation
az provider register --namespace Microsoft.ExtendedLocation --wait
az provider register --namespace Microsoft.Web --wait
az provider register --namespace Microsoft.KubernetesConfiguration --wait
az extension remove --name appservice-kube
az extension add --upgrade --yes --name appservice-kube
```

## Create a connected cluster

### NOTE

This tutorial uses [Azure Kubernetes Service \(AKS\)](#) to provide concrete instructions for setting up an environment from scratch. However, for a production workload, you will likely not want to enable Azure Arc on an AKS cluster as it is already managed in Azure. The steps below will help you get started understanding the service, but for production deployments, they should be viewed as illustrative, not prescriptive. See [Quickstart: Connect an existing Kubernetes cluster to Azure Arc](#) for general instructions on creating an Azure Arc-enabled Kubernetes cluster.

1. Create a cluster in Azure Kubernetes Service with a public IP address. Replace <group-name> with the resource group name you want.
  - [bash](#)
  - [PowerShell](#)

```
aksClusterGroupName="" # Name of resource group for the AKS cluster
aksName="${aksClusterGroupName}-aks" # Name of the AKS cluster
resourceLocation="eastus" # "eastus" or "westeurope"

az group create -g $aksClusterGroupName -l $resourceLocation
az aks create --resource-group $aksClusterGroupName --name $aksName --enable-aad --generate-ssh-keys
infra_rg=$(az aks show --resource-group $aksClusterGroupName --name $aksName --output tsv --query nodeResourceGroup)
```

2. Get the [kubeconfig](#) file and test your connection to the cluster. By default, the kubeconfig file is saved to `~/.kube/config`.

```
az aks get-credentials --resource-group $aksClusterGroupName --name $aksName --admin

kubectl get ns
```

3. Create a resource group to contain your Azure Arc resources. Replace `<group-name>` with the resource group name you want.

- [bash](#)
- [PowerShell](#)

```
groupName="" # Name of resource group for the connected cluster

az group create -g $groupName -l $resourceLocation
```

4. Connect the cluster you created to Azure Arc.

- [bash](#)
- [PowerShell](#)

```
clusterName="${groupName}-cluster" # Name of the connected cluster resource

az connectedk8s connect --resource-group $groupName --name $clusterName
```

5. Validate the connection with the following command. It should show the `provisioningState` property as `Succeeded`. If not, run the command again after a minute.

```
az connectedk8s show --resource-group $groupName --name $clusterName
```

## Create a Log Analytics workspace

While a [Log Analytic workspace](#) is not required to run App Service in Azure Arc, it's how developers can get application logs for their apps that are running in the Azure Arc-enabled Kubernetes cluster.

1. For simplicity, create the workspace now.

- [bash](#)
- [PowerShell](#)

```
workspaceName="$groupName-workspace" # Name of the Log Analytics workspace

az monitor log-analytics workspace create \
--resource-group $groupName \
--workspace-name $workspaceName
```

- Run the following commands to get the encoded workspace ID and shared key for an existing Log Analytics workspace. You need them in the next step.

- bash
- PowerShell

```
logAnalyticsWorkspaceId=$(az monitor log-analytics workspace show \
--resource-group $groupName \
--workspace-name $workspaceName \
--query customerId \
--output tsv)
logAnalyticsWorkspaceIdEnc=$(printf %s $logAnalyticsWorkspaceId | base64 -w0) # Needed for the next step
logAnalyticsKey=$(az monitor log-analytics workspace get-shared-keys \
--resource-group $groupName \
--workspace-name $workspaceName \
--query primarySharedKey \
--output tsv)
logAnalyticsKeyEnc=$(printf %s $logAnalyticsKey | base64 -w0) # Needed for the next step
```

## Install the App Service extension

- Set the following environment variables for the desired name of the [App Service extension](#), the cluster namespace in which resources should be provisioned, and the name for the App Service Kubernetes environment. Choose a unique name for <kube-environment-name>, because it will be part of the domain name for app created in the App Service Kubernetes environment.

- bash
- PowerShell

```
extensionName="appservice-ext" # Name of the App Service extension
namespace="appservice-ns" # Namespace in your cluster to install the extension and provision resources
kubeEnvironmentName="" # Name of the App Service Kubernetes environment resource
```

- Install the App Service extension to your Azure Arc-connected cluster, with Log Analytics enabled. Again, while Log Analytics is not required, you can't add it to the extension later, so it's easier to do it now.

- bash
- PowerShell

```

az k8s-extension create \
--resource-group $groupName \
--name $extensionName \
--cluster-type connectedClusters \
--cluster-name $clusterName \
--extension-type 'Microsoft.Web.Appservice' \
--release-train stable \
--auto-upgrade-minor-version true \
--scope cluster \
--release-namespace $namespace \
--configuration-settings "Microsoft.CustomLocation.ServiceAccount=default" \
--configuration-settings "appsNamespace=${namespace}" \
--configuration-settings "clusterName=${kubeEnvironmentName}" \
--configuration-settings "keda.enabled=true" \
--configuration-settings "buildService.storageClassName=default" \
--configuration-settings "buildService.storageAccessMode=ReadWriteOnce" \
--configuration-settings "customConfigMap=${namespace}/kube-environment-config" \
--configuration-settings "envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-
resource-group=${aksClusterGroupName}" \
--configuration-settings "logProcessor.appLogs.destination=log-analytics" \
--configuration-protected-settings
"logProcessor.appLogs.logAnalyticsConfig.customerId=${logAnalyticsWorkspaceIdEnc}" \
--configuration-protected-settings
"logProcessor.appLogs.logAnalyticsConfig.sharedKey=${logAnalyticsKeyEnc}"

```

#### NOTE

To install the extension without Log Analytics integration, remove the last three `--configuration-settings` parameters from the command.

The following table describes the various `--configuration-settings` parameters when running the command:

PARAMETER	DESCRIPTION
<code>Microsoft.CustomLocation.ServiceAccount</code>	The service account that should be created for the custom location that will be created. It is recommended that this be set to the value <code>default</code> .
<code>appsNamespace</code>	The namespace to provision the app definitions and pods. <b>Must</b> match that of the extension release namespace.
<code>clusterName</code>	The name of the App Service Kubernetes environment that will be created against this extension.
<code>keda.enabled</code>	Whether <b>KEDA</b> should be installed on the Kubernetes cluster. Accepts <code>true</code> or <code>false</code> .
<code>buildService.storageClassName</code>	The <b>name of the storage class</b> for the build service to store build artifacts. A value like <code>default</code> specifies a class named <code>default</code> , and not <b>any class that is marked as default</b> . Default is a valid storage class for AKS and AKS HCI but it may not be for other distributions/platforms.
<code>buildService.storageAccessMode</code>	The <b>access mode</b> to use with the named storage class above. Accepts <code>ReadWriteOnce</code> or <code>ReadWriteMany</code> .

PARAMETER	DESCRIPTION
<code>customConfigMap</code>	The name of the config map that will be set by the App Service Kubernetes environment. Currently, it must be <code>&lt;namespace&gt;/kube-environment-config</code> , replacing <code>&lt;namespace&gt;</code> with the value of <code>appsNamespace</code> above.
<code>envoy.annotations.service.beta.kubernetes.io/azure-load-balancer-resource-group</code>	The name of the resource group in which the Azure Kubernetes Service cluster resides. Valid and required only when the underlying cluster is Azure Kubernetes Service.
<code>logProcessor.appLogs.destination</code>	Optional. Accepts <code>log-analytics</code> or <code>none</code> , choosing <code>none</code> disables platform logs.
<code>logProcessor.appLogs.logAnalyticsConfig.customerId</code>	Required only when <code>logProcessor.appLogs.destination</code> is set to <code>log-analytics</code> . The base64-encoded Log analytics workspace ID. This parameter should be configured as a protected setting.
<code>logProcessor.appLogs.logAnalyticsConfig.sharedKey</code>	Required only when <code>logProcessor.appLogs.destination</code> is set to <code>log-analytics</code> . The base64-encoded Log analytics workspace shared key. This parameter should be configured as a protected setting.

3. Save the `id` property of the App Service extension for later.

- [bash](#)
- [PowerShell](#)

```
extensionId=$(az k8s-extension show \
 --cluster-type connectedClusters \
 --cluster-name $clusterName \
 --resource-group $groupName \
 --name $extensionName \
 --query id \
 --output tsv)
```

4. Wait for the extension to fully install before proceeding. You can have your terminal session wait until this complete by running the following command:

```
az resource wait --ids $extensionId --custom "properties.installState!='Pending'" --api-version
"2020-07-01-preview"
```

You can use `kubectl` to see the pods that have been created in your Kubernetes cluster:

```
kubectl get pods -n $namespace
```

You can learn more about these pods and their role in the system from [Pods created by the App Service extension](#).

## Create a custom location

The [custom location](#) in Azure is used to assign the App Service Kubernetes environment.

1. Set the following environment variables for the desired name of the custom location and for the ID of the Azure Arc-connected cluster.

- [bash](#)
- [PowerShell](#)

```
customLocationName="my-custom-location" # Name of the custom location

connectedClusterId=$(az connectedk8s show --resource-group $groupName --name $clusterName --query id
--output tsv)
```

2. Create the custom location:

- [bash](#)
- [PowerShell](#)

```
az customlocation create \
--resource-group $groupName \
--name $customLocationName \
--host-resource-id $connectedClusterId \
--namespace $namespace \
--cluster-extension-ids $extensionId
```

3. Validate that the custom location is successfully created with the following command. The output should show the `provisioningState` property as `Succeeded`. If not, run it again after a minute.

```
az customlocation show --resource-group $groupName --name $customLocationName
```

4. Save the custom location ID for the next step.

- [bash](#)
- [PowerShell](#)

```
customLocationId=$(az customlocation show \
--resource-group $groupName \
--name $customLocationName \
--query id \
--output tsv)
```

## Create the App Service Kubernetes environment

Before you can start creating apps on the custom location, you need an [App Service Kubernetes environment](#).

1. Create the App Service Kubernetes environment:

- [bash](#)
- [PowerShell](#)

```
az appservice kube create \
--resource-group $groupName \
--name $kubeEnvironmentName \
--custom-location $customLocationId \

```

2. Validate that the App Service Kubernetes environment is successfully created with the following command. The output should show the `provisioningState` property as `Succeeded`. If not, run it again after a minute.

```
az appservice kube show --resource-group $groupName --name $kubeEnvironmentName
```

## Next steps

- [Quickstart: Create a web app on Azure Arc](#)
- [Create your first function on Azure Arc](#)
- [Create your first logic app on Azure Arc](#)

# Enable virtual network integration in Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

Through integrating with an Azure virtual network (VNet) from your [App Service app](#), you can reach private resources from your app within the virtual network. The VNet Integration feature has two variations:

- Regional VNet integration: Connect to Azure virtual networks in the same region. You must have a dedicated subnet in the VNet you're integrating with.
- Gateway-required VNet integration: When you connect directly to VNet in other regions or to a classic virtual network in the same region, you must use the gateway-required VNet integration.

This how-to article will describe how to set up regional VNet integration.

## Prerequisites

The VNet Integration requires:

- An App Service pricing tier [supporting VNet integration](#).
- A virtual network in the same region with an empty subnet.

The subnet must be delegated to Microsoft.Web/serverFarms. If the delegation isn't done before integration, the provisioning process will configure this delegation. The subnet must be allocated an IPv4 /28 block (16 addresses). It is actually recommended to have a minimum of 64 addresses (IPv4 /26 block) to allow for maximum horizontal scale.

## Configure in the Azure portal

1. Go to the **Networking** UI in the App Service portal. Under **Outbound Traffic**, select **VNet integration**.
2. Select **Add VNet**.

The screenshot shows the 'VNet Integration' configuration page in the Azure portal. At the top, there's a breadcrumb trail: Home > vnet-integration-setup > vnet-integration-webapp >. Below it is a header with 'VNet Integration' and a 'vnets-integration-webapp' link. There are 'Disconnect' and 'Refresh' buttons. The main area is titled 'VNet Configuration' with a sub-section 'Securely access resources available in or through your Azure VNet. [Learn more](#)'. A '+ Add VNet' button is present. The 'VNet Details' section shows 'VNet NAME' and 'LOCATION' both as 'Not Configured'. The 'VNet Address Space' section shows 'Start Address' as 'Not Configured' and 'End Address' as 'Not Configured'.

3. The drop-down list contains all of the virtual networks in your subscription in the same region.

The screenshot shows the Azure portal interface. On the left, the 'VNet Configuration' blade is open for a web app named 'vnet-integration-webapp'. It displays 'VNet Details' (VNet NAME: Not Configured, LOCATION: Not Configured) and 'VNet Address Space' (Start Address: Not Configured, End Address: Not Configured). On the right, the 'Add VNet Integration' dialog is open, prompting the user to connect to a Classic VNet. It includes fields for 'Subscription' (set to 'Azure Subscription') and 'Virtual Network' (a dropdown menu with options: 'Same region (West Europe)', 'integration-vnet (West Europe)', 'Other regions (requires a Virtual Network Gateway configured with Point to Site VPN)', and 'remote-vnet (North Europe)'). A blue 'OK' button is at the bottom of the dialog.

- Select an empty pre-existing subnet or create a new subnet.

During the integration, your app is restarted. When integration is finished, you'll see details on the VNet you're integrated with.

## Configure with Azure CLI

You can also configure VNet integration using Azure CLI:

```
az webapp vnet-integration add --resource-group <group-name> --name <app-name> --vnet <vnet-name> --subnet <subnet-name>
```

### NOTE

The command will check if subnet is delegated to Microsoft.Web/serverFarms and apply the necessary delegation if this is not configured. If this has already been configured, and you do not have permissions to check this, or the virtual network is in another subscription, you can use the `--skip-delegation-check` parameter to bypass the validation.

## Configure with Azure PowerShell

```
Parameters
$siteName = '<app-name>'
$resourceGroupName = '<group-name>'
$vNetName = '<vnet-name>'
$integrationSubnetName = '<subnet-name>'
$subscriptionId = '<subscription-guid>'

Configure VNet Integration
$subnetResourceId =
"/subscriptions/$subscriptionId/resourceGroups/$resourceGroupName/providers/Microsoft.Network/virtualNetworks/$vNetName/subnets/$integrationSubnetName"
$webApp = Get-AzResource -ResourceType Microsoft.Web/sites -ResourceGroupName $resourceGroupName -ResourceName $siteName
$webApp.Properties.virtualNetworkSubnetId = $subnetResourceId
$webApp | Set-AzResource -Force
```

## Next steps

- [Configure VNet integration routing](#)
- [General Networking overview](#)

# Manage Azure App Service virtual network integration routing

11/2/2021 • 2 minutes to read • [Edit Online](#)

When configuring application routing, you can either route all traffic or only private traffic (also known as [RFC1918](#) traffic) into your Azure virtual network (VNet). This how-to article will describe how to configure application routing.

## Prerequisites

Your app is already integrated using the regional VNet integration feature.

## Configure in the Azure portal

You can use the following steps to disable Route All in your app through the portal:

The screenshot shows the Azure portal interface for managing VNet integration. At the top, there's a breadcrumb navigation: Home > vnet-integration-webapp >. Below it is a header with the title "VNet Integration" and a "vnet-integration-webapp" subtitle. There are three buttons: "Disconnect", "Refresh", and "Troubleshoot".

Under the "VNet Configuration" section, there's a sub-header "Securely access resources available in or through your Azure VNet. [Learn more](#)".

The main configuration area has a "Route All" setting, which is currently "Enabled". A note below it says: "When Route All is enabled, all outbound traffic from the app will be affected by networking configurations. [Learn more](#)".

Below this, under "VNet Details", there are two entries: "VNet NAME" set to "integration-vnet" and "LOCATION" set to "West Europe".

1. Go to the **Networking > VNet integration** UI in your app portal.
2. Set **Route All** to Disabled.

**Disable VNet Route All configuration.**

Are you sure you want to disable VNet Route All?

Yes  No

**Route All**  Disabled  
 When Route All is enabled, all outbound traffic from the app will be affected by networking configurations. [Learn more](#)

**VNet Details**

VNet NAME	integration-vnet
LOCATION	West Europe

3. Select **Yes** to confirm.

## Configure with Azure CLI

You can also configure Route All using Azure CLI (the minimum `az version` required is 2.27.0):

```
az webapp config set --resource-group <group-name> --name <app-name> --vnet-route-all-enabled [true|false]
```

## Configure with Azure PowerShell

```
Parameters
$siteName = '<app-name>'
$resourceGroupName = '<group-name>'

Configure VNet Integration
$webApp = Get-AzResource -ResourceType Microsoft.Web/sites -ResourceGroupName $resourceGroupName -
 ResourceName $siteName
Set-AzResource -ResourceId ($webApp.Id + "/config/web") -Properties @{ vnetRouteAllEnabled = $true } -Force
```

## Next steps

- [Enable VNet integration](#)
- [General Networking overview](#)

# Move an App Service resource to another region

11/2/2021 • 2 minutes to read • [Edit Online](#)

This article describes how to move App Service resources to a different Azure region. You might move your resources to another region for a number of reasons. For example, to take advantage of a new Azure region, to deploy features or services available in specific regions only, to meet internal policy and governance requirements, or in response to capacity planning requirements.

App Service resources are region-specific and can't be moved across regions. You must create a copy of your existing App Service resources in the target region, then move your content over to the new app. If your source app uses a custom domain, you can [migrate it to the new app in the target region](#) when you're finished.

To make copying your app easier, you can [clone an individual App Service app](#) into an App Service plan in another region, but it does have [limitations](#), especially that it doesn't support Linux apps.

## Prerequisites

- Make sure that the App Service app is in the Azure region from which you want to move.
- Make sure that the target region supports App Service and any related service, whose resources you want to move.

## Prepare

Identify all the App Service resources that you're currently using. For example:

- App Service apps
- [App Service plans](#)
- [Deployment slots](#)
- [Custom domains purchased in Azure](#)
- [TLS/SSL certificates](#)
- [Azure Virtual Network integration](#)
- [Hybrid connections.](#)
- [Managed identities](#)
- [Backup settings](#)

Certain resources, such as imported certificates or hybrid connections, contain integration with other Azure services. For information on how to move those resources across regions, see the documentation for the respective services.

## Move

1. [Create a back up of the source app.](#)
2. [Create an app in a new App Service plan, in the target region.](#)
3. [Restore the back up in the target app](#)
4. If you use a custom domain, [bind it preemptively to the target app](#) with `awverify`. and [enable the domain in the target app](#).
5. Configure everything else in your target app to be the same as the source app and verify your configuration.
6. When you're ready for the custom domain to point to the target app, [remap the domain name](#).

## Clean up source resources

Delete the source app and App Service plan. [An App Service plan in the non-free tier carries a charge, even if no app is running in it.](#)

## Next steps

[Azure App Service App Cloning Using PowerShell](#)

# Move resources to a new resource group or subscription

11/2/2021 • 13 minutes to read • [Edit Online](#)

This article shows you how to move Azure resources to either another Azure subscription or another resource group under the same subscription. You can use the Azure portal, Azure PowerShell, Azure CLI, or the REST API to move resources.

Both the source group and the target group are locked during the move operation. Write and delete operations are blocked on the resource groups until the move completes. This lock means you can't add, update, or delete resources in the resource groups. It doesn't mean the resources are frozen. For example, if you move an Azure SQL logical server and its databases to a new resource group or subscription, applications that use the databases experience no downtime. They can still read and write to the databases. The lock can last for a maximum of four hours, but most moves complete in much less time.

Moving a resource only moves it to a new resource group or subscription. It doesn't change the location of the resource.

## Changed resource ID

When you move a resource, you change its resource ID. The standard format for a resource ID is

/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/{resourceProviderNamespace}/{resourceType}/{resourceName}

. When you move a resource to a new resource group or subscription, you change one or more values in that path.

If you use the resource ID anywhere, you'll need to change that value. For example, if you have a [custom dashboard](#) in the portal that references a resource ID, you'll need to update that value. Look for any scripts or templates that need to be updated for the new resource ID.

## Checklist before moving resources

There are some important steps to do before moving a resource. By verifying these conditions, you can avoid errors.

1. The resources you want to move must support the move operation. For a list of which resources support move, see [Move operation support for resources](#).
2. Some services have specific limitations or requirements when moving resources. If you're moving any of the following services, check that guidance before moving.
  - If you're using Azure Stack Hub, you can't move resources between groups.
  - [App Services move guidance](#)
  - [Azure DevOps Services move guidance](#)
  - [Classic deployment model move guidance](#) - Classic Compute, Classic Storage, Classic Virtual Networks, and Cloud Services
  - [Networking move guidance](#)
  - [Recovery Services move guidance](#)
  - [Virtual Machines move guidance](#)
  - To move an Azure subscription to a new management group, see [Move subscriptions](#).
3. If you move a resource that has an Azure role assigned directly to the resource (or a child resource), the role assignment isn't moved and becomes orphaned. After the move, you must re-create the role assignment. Eventually, the orphaned role assignment is automatically removed, but we recommend removing the role assignment before the move.

For information about how to manage role assignments, see [List Azure role assignments](#) and [Assign Azure roles](#).

4. The source and destination subscriptions must be active. If you have trouble enabling an account that has been disabled, [create an Azure support request](#). Select **Subscription Management** for the issue type.

5. The source and destination subscriptions must exist within the same [Azure Active Directory tenant](#). To check that both subscriptions have the same tenant ID, use Azure PowerShell or Azure CLI.

For Azure PowerShell, use:

```
(Get-AzSubscription -SubscriptionName <your-source-subscription>).TenantId
(Get-AzSubscription -SubscriptionName <your-destination-subscription>).TenantId
```

For Azure CLI, use:

```
az account show --subscription <your-source-subscription> --query tenantId
az account show --subscription <your-destination-subscription> --query tenantId
```

If the tenant IDs for the source and destination subscriptions aren't the same, use the following methods to reconcile the tenant IDs:

- [Transfer ownership of an Azure subscription to another account](#)
- [How to associate or add an Azure subscription to Azure Active Directory](#)

6. The destination subscription must be registered for the resource provider of the resource being moved. If not, you receive an error stating that the **subscription is not registered for a resource type**. You might see this error when moving a resource to a new subscription, but that subscription has never been used with that resource type.

For PowerShell, use the following commands to get the registration status:

```
Set-AzContext -Subscription <destination-subscription-name-or-id>
Get-AzResourceProvider -ListAvailable | Select-Object ProviderNamespace, RegistrationState
```

To register a resource provider, use:

```
Register-AzResourceProvider -ProviderNamespace Microsoft.Batch
```

For Azure CLI, use the following commands to get the registration status:

```
az account set -s <destination-subscription-name-or-id>
az provider list --query "[].{Provider:namespace, Status:registrationState}" --out table
```

To register a resource provider, use:

```
az provider register --namespace Microsoft.Batch
```

7. The account moving the resources must have at least the following permissions:

- **Microsoft.Resources/subscriptions/resourceGroups/moveResources/action** on the source resource group.
- **Microsoft.Resources/subscriptions/resourceGroups/write** on the destination resource group.

8. Before moving the resources, check the subscription quotas for the subscription you're moving the resources to. If moving the resources means the subscription will exceed its limits, you need to review whether you can request an increase in the quota. For a list of limits and how to request an increase, see [Azure subscription and service limits, quotas, and constraints](#).

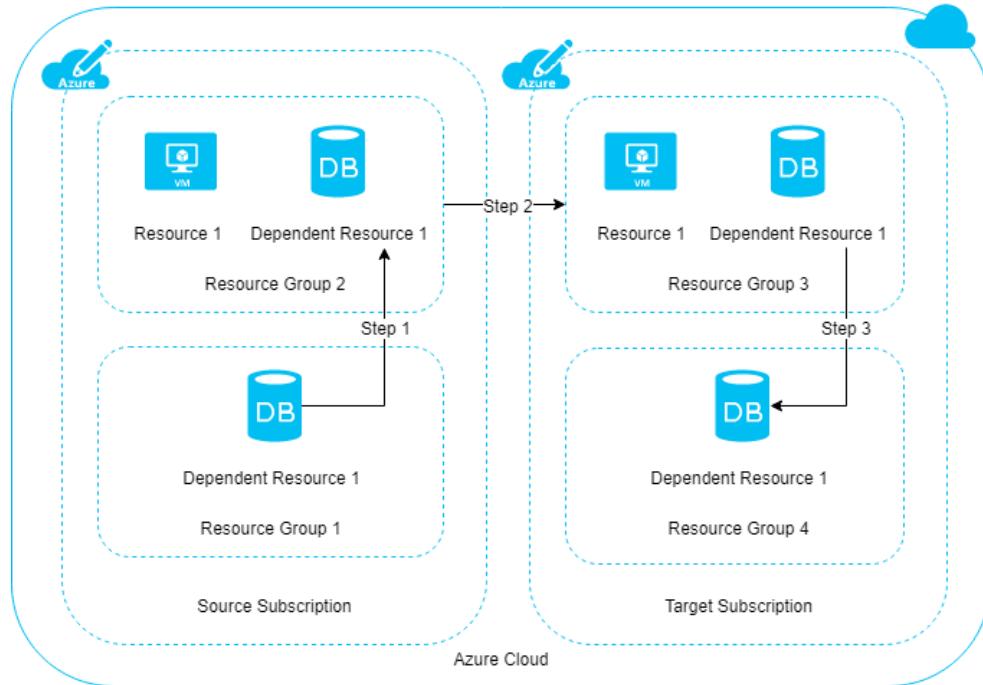
9. **For a move across subscriptions, the resource and its dependent resources must be located in the same resource group and they must be moved together.** For example, a VM with managed disks would require the VM and the managed disks to be moved together, along with other dependent resources.

If you're moving a resource to a new subscription, check to see whether the resource has any dependent resources, and whether they're located in the same resource group. If the resources aren't in the same resource group, check to see whether the resources can be combined into the same resource group. If so, bring all these resources into the same resource group by using a move operation across resource groups.

For more information, see [Scenario for move across subscriptions](#).

## Scenario for move across subscriptions

Moving resources from one subscription to another is a three-step process:



For illustration purposes, we have only one dependent resource.

- Step 1: If dependent resources are distributed across different resource groups, first move them into one resource group.
- Step 2: Move the resource and dependent resources together from the source subscription to the target subscription.
- Step 3: Optionally, redistribute the dependent resources to different resource groups within the target subscription.

## Use the portal

To move resources, select the resource group that contains those resources.

Select the resources you want to move. To move all of the resources, select the checkbox at the top of list. Or, select resources individually.

Microsoft Azure

Home > Resource groups >

**sourceGroup** Resource group

Search (Ctrl+ /) Overview Activity log Access control (IAM) Tags Events

Deployments Security Policies Properties Locks

Cost analysis Cost alerts (preview)

Create Edit columns Delete resource group Refresh

Essentials

Subscription (change) : Documentation Testing 1  
Subscription ID :  
Tags (change) : Click here to add tags

Filter for any field... Type == all Location == all

Showing 1 to 6 of 6 records.  Show hidden types

Name
exampleVM1
exampleVM1-ip
exampleVM1-nsg
examplevm1920
exampleVM1_OsDisk_1_38427735e90a4270a5888d64e37065de
sourceGroup-vnet

The 'Name' column header is highlighted with a red box.

Select the **Move** button.

Create Edit columns Delete resource group Refresh Export to CSV Open query Assign tags Move Delete Export

Essentials

Subscription (change) : Documentation Testing 1  
Subscription ID :  
Tags (change) : Click here to add tags

Deployments : 1 Subscriptions : 1 Locations : West

Move to another resource group  
Move to another subscription  
Move to another region

This button gives you three options:

- Move to a new resource group.
- Move to a new subscription.
- Move to a new region. To change regions, see [Move resources across regions \(from resource group\)](#).

Select whether you're moving the resources to a new resource group or a new subscription.

The source resource group is automatically set. Specify the destination resource group. If you're moving to a new subscription, also specify the subscription. Select **Next**.

Home > sourceGroup >

## Move resources ...

sourceGroup

1 Source + target    2 Resources to move    3 Review

To move a resource, select a source and a destination. The source and destination resource groups will both be locked during the move. [Learn more](#)

### Source

Subscription                          Documentation Testing 1

Resource group                        sourceGroup

### Target

Subscription                                  Documentation Testing 1

Resource group \*                         destinationGroup  
Create new

Previous

Next

The portal validates that the resources can be moved. Wait for validation to complete.

Home > sourceGroup >

## Move resources ...

sourceGroup

1 Source + target    2 Resources to move    3 Review

Checking whether these resources can be moved. This might take a few minutes.

+ Add resources

X Remove from the move list

Name	Type	Resource type	Validation status
exampleVM1	Virtual machine	microsoft.compute/virtualmachines	Pending validation
exampleVM1-ip	Public IP address	microsoft.network/publicipaddresses	Pending validation
exampleVM1-nsg	Network security group	microsoft.network/networksecuritygroups	Pending validation
examplevm1920	Network interface	microsoft.network/networkinterfaces	Pending validation
exampleVM1_OsDisk_1_38427735e90a427l	Disk	microsoft.compute/disks	Pending validation
sourceGroup-vnet	Virtual network	microsoft.network/virtualnetworks	Pending validation

When validation completes successfully, select **Next**.

Acknowledge that you need to update tools and scripts for these resources. To start moving the resources, select **Move**.

Home > sourceGroup >

## Move resources

...

sourceGroup

1 Source + target    2 Resources to move    3 Review

### Selection summary

Source subscription	Documentation Testing 1
Source resource group	sourceGroup
Target subscription	Documentation Testing 1
Target resource group	destinationGroup
Number of resources to move	6

I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs

Previous

Move

When the move has completed, you're notified of the result.



## Notifications

X

[More events in the activity log →](#)

[Dismiss all](#) ▾

Moving resources complete

X

Successfully moved 6 resources from resource group 'sourceGroup' in subscription 'Documentation Testing 1' to resource group 'destinationGroup' in subscription 'Documentation Testing 1'

[Feedback](#)

[Related events](#)

12 minutes ago

## Use Azure PowerShell

### Validate

To test your move scenario without actually moving the resources, use the [Invoke-AzResourceAction](#) command.

Use this command only when you need to predetermine the results. To run this operation, you need the:

- resource ID of the source resource group
- resource ID of the target resource group
- resource ID of each resource to move

```
Invoke-AzResourceAction -Action validateMoveResources `
-ResourceId "/subscriptions/{subscription-id}/resourceGroups/{source-rg}" `
-Parameters @{ resources= @("/subscriptions/{subscription-id}/resourceGroups/{source-
rg}/providers/{resource-provider}/{resource-type}/{resource-name}", "/subscriptions/{subscription-
id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-type}/{resource-name}",
"/subscriptions/{subscription-id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-
type}/{resource-name}");targetResourceGroup = '/subscriptions/{subscription-id}/resourceGroups/{destination-
rg}' }
```

If validation passes, you see no output.

If validation fails, you see an error message describing why the resources can't be moved.

## Move

To move existing resources to another resource group or subscription, use the [Move-AzResource](#) command. The following example shows how to move several resources to a new resource group.

```
$webapp = Get-AzResource -ResourceGroupName OldRG -ResourceName ExampleSite
$plan = Get-AzResource -ResourceGroupName OldRG -ResourceName ExamplePlan
Move-AzResource -DestinationResourceGroupName NewRG -ResourceId $webapp.ResourceId, $plan.ResourceId
```

To move to a new subscription, include a value for the `DestinationSubscriptionId` parameter.

## Use Azure CLI

### Validate

To test your move scenario without actually moving the resources, use the [az resource invoke-action](#) command. Use this command only when you need to predetermine the results. To run this operation, you need the:

- resource ID of the source resource group
- resource ID of the target resource group
- resource ID of each resource to move

In the request body, use `\\"` to escape double quotes.

```
az resource invoke-action --action validateMoveResources \
--ids "/subscriptions/{subscription-id}/resourceGroups/{source-rg}" \
--request-body "{ \"resources\": [\"/subscriptions/{subscription-id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-type}/{resource-name}\", \"/subscriptions/{subscription-id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-type}/{resource-name}\", \
\"/subscriptions/{subscription-id}/resourceGroups/{source-rg}/providers/{resource-provider}/{resource-type}/{resource-name}\"], \"targetResourceGroup\": \"/subscriptions/{subscription-id}/resourceGroups/{destination-rg}\\" }"
```

If validation passes, you see:

```
{} Finished ..
```

If validation fails, you see an error message describing why the resources can't be moved.

## Move

To move existing resources to another resource group or subscription, use the [az resource move](#) command. Provide the resource IDs of the resources to move. The following example shows how to move several resources to a new resource group. In the `--ids` parameter, provide a space-separated list of the resource IDs to move.

```
webapp=$(az resource show -g OldRG -n ExampleSite --resource-type "Microsoft.Web/sites" --query id --output tsv)
plan=$(az resource show -g OldRG -n ExamplePlan --resource-type "Microsoft.Web/serverfarms" --query id --output tsv)
az resource move --destination-group newgroup --ids $webapp $plan
```

To move to a new subscription, provide the `--destination-subscription-id` parameter.

## Use REST API

### Validate

The [validate move operation](#) lets you test your move scenario without actually moving the resources. Use this operation to check if the move will succeed. Validation is automatically called when you send a move request. Use this operation only when you need to predetermine the results. To run this operation, you need the:

- name of the source resource group
- resource ID of the target resource group
- resource ID of each resource to move
- the [access token](#) for your account

Send the following request:

```
POST https://management.azure.com/subscriptions/<subscription-id>/resourceGroups/<source-group>/validateMoveResources?api-version=2019-05-10
Authorization: Bearer <access-token>
Content-type: application/json
```

With a request body:

```
{
 "resources": ["<resource-id-1>", "<resource-id-2>"],
 "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

If the request is formatted correctly, the operation returns:

```
Response Code: 202
cache-control: no-cache
pragma: no-cache
expires: -1
location: https://management.azure.com/subscriptions/<subscription-id>/operationresults/<operation-id>?api-version=2018-02-01
retry-after: 15
...
...
```

The 202 status code indicates the validation request was accepted, but it hasn't yet determined if the move operation will succeed. The `location` value contains a URL that you use to check the status of the long-running operation.

To check the status, send the following request:

```
GET <location-url>
Authorization: Bearer <access-token>
```

While the operation is still running, you continue to receive the 202 status code. Wait the number of seconds indicated in the `retry-after` value before trying again. If the move operation validates successfully, you receive the 204 status code. If the move validation fails, you receive an error message, such as:

```
{"error":{"code":"ResourceMoveProviderValidationFailed","message":"><message>...}}
```

## Move

To move existing resources to another resource group or subscription, use the [Move resources](#) operation.

```
POST https://management.azure.com/subscriptions/{source-subscription-id}/resourcegroups/{source-resource-group-name}/moveResources?api-version={api-version}
```

In the request body, you specify the target resource group and the resources to move.

```
{
 "resources": ["<resource-id-1>", "<resource-id-2>"],
 "targetResourceGroup": "/subscriptions/<subscription-id>/resourceGroups/<target-group>"
}
```

## Frequently asked questions

**Question:** My resource move operation, which usually takes a few minutes, has been running for almost an hour. Is there something wrong?

Moving a resource is a complex operation that has different phases. It can involve more than just the resource provider of the resource you're trying to move. Because of the dependencies between resource providers, Azure Resource Manager allows 4 hours for the operation to complete. This time period gives resource providers a chance to recover from transient issues. If your move request is within the four-hour period, the operation keeps trying to complete and may still succeed. The source and destination resource groups are locked during this time to avoid consistency issues.

**Question:** Why is my resource group locked for four hours during resource move?

A move request is allowed a maximum of four hours to complete. To prevent modifications on the resources being moved, both the source and destination resource groups are locked during the resource move.

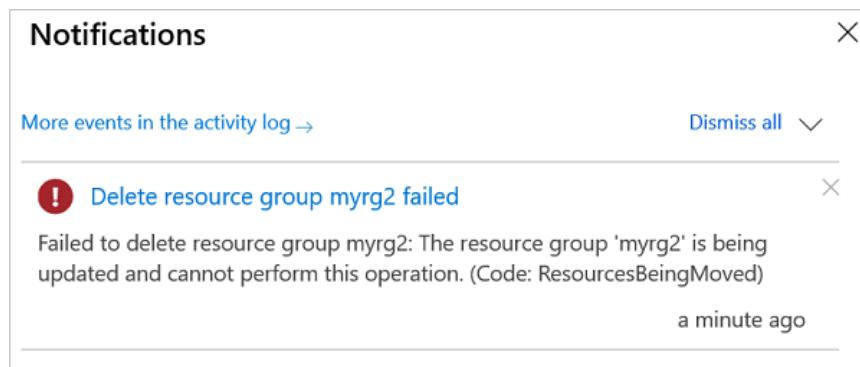
There are two phases in a move request. In the first phase, the resource is moved. In the second phase, notifications are sent to other resource providers that are dependent on the resource being moved. A resource group can be locked for the entire four hours when a resource provider fails either phase. During the allowed time, Resource Manager retries the failed step.

If a resource can't be moved within four hours, Resource Manager unlocks both resource groups. Resources that were successfully moved are in the destination resource group. Resources that failed to move are left the source resource group.

**Question:** What are the implications of the source and destination resource groups being locked during the resource move?

The lock prevents you from deleting either resource group, creating a new resource in either resource group, or deleting any of the resources involved in the move.

The following image shows an error message from the Azure portal when a user tries to delete a resource group that is part of an ongoing move.



**Question:** What does the error code "MissingMoveDependentResources" mean?

When moving a resource, its dependent resources must either exist in the destination resource group or subscription, or be included in the move request. You get the MissingMoveDependentResources error code when a dependent resource doesn't meet this requirement. The error message has details about the dependent resource that needs to be included in the move request.

For example, moving a virtual machine could require moving seven resource types with three different resource providers. Those resource providers and types are:

- Microsoft.Compute
  - virtualMachines
  - disks
- Microsoft.Network
  - networkInterfaces
  - publicIPAddresses

- networkSecurityGroups
- virtualNetworks
- Microsoft.Storage
  - storageAccounts

Another common example involves moving a virtual network. You may have to move several other resources associated with that virtual network. The move request could require moving public IP addresses, route tables, virtual network gateways, network security groups, and others.

**Question: What does the error code "RequestDisallowedByPolicy" mean?**

Resource Manager validates your move request before attempting the move. This validation includes checking policies defined on the resources involved in the move. For example, if you're attempting to move a key vault but your organization has a policy to deny the creation of a key vault in the target resource group, validation fails and the move is blocked. The returned error code is **RequestDisallowedByPolicy**.

For more information about policies, see [What is Azure Policy?](#).

**Question: Why can't I move some resources in Azure?**

Currently, not all resources in Azure support move. For a list of resources that support move, see [Move operation support for resources](#).

**Question: How many resources can I move in a single operation?**

When possible, break large moves into separate move operations. Resource Manager immediately returns an error when there are more than 800 resources in a single operation. However, moving less than 800 resources may also fail by timing out.

**Question: What is the meaning of the error that a resource isn't in succeeded state?**

When you get an error message that indicates a resource can't be moved because it isn't in a succeeded state, it may actually be a dependent resource that is blocking the move. Typically, the error code is **MoveCannotProceedWithResourcesNotInSucceededState**.

If the source or target resource group contains a virtual network, the states of all dependent resources for the virtual network are checked during the move. The check includes those resources directly and indirectly dependent on the virtual network. If any of those resources are in a failed state, the move is blocked. For example, if a virtual machine that uses the virtual network has failed, the move is blocked. The move is blocked even when the virtual machine isn't one of the resources being moved and isn't in one of the resource groups for the move.

When you receive this error, you have two options. Either move your resources to a resource group that doesn't have a virtual network, or [contact support](#).

## Next steps

For a list of which resources support move, see [Move operation support for resources](#).

# Run background tasks with WebJobs in Azure App Service

11/2/2021 • 7 minutes to read • [Edit Online](#)

Deploy WebJobs by using the [Azure portal](#) to upload an executable or script. You can run background tasks in the Azure App Service.

If instead of the Azure App Service you are using Visual Studio 2019 to develop and deploy WebJobs, see [Deploy WebJobs using Visual Studio](#).

## Overview

WebJobs is a feature of [Azure App Service](#) that enables you to run a program or script in the same instance as a web app, API app, or mobile app. There is no additional cost to use WebJobs.

You can use the Azure WebJobs SDK with WebJobs to simplify many programming tasks. WebJobs is not yet supported for App Service on Linux. For more information, see [What is the WebJobs SDK](#).

Azure Functions provides another way to run programs and scripts. For a comparison between WebJobs and Functions, see [Choose between Flow, Logic Apps, Functions, and WebJobs](#).

## WebJob types

The following table describes the differences between *continuous* and *triggered* WebJobs.

CONTINUOUS	TRIGGERED
Starts immediately when the WebJob is created. To keep the job from ending, the program or script typically does its work inside an endless loop. If the job does end, you can restart it. Typically used with WebJobs SDK.	Starts only when triggered manually or on a schedule.
Runs on all instances that the web app runs on. You can optionally restrict the WebJob to a single instance.	Runs on a single instance that Azure selects for load balancing.
Supports remote debugging.	Doesn't support remote debugging.
Code is deployed under <code>\site\wwwroot\app_data\Jobs\Continuous</code> .	Code is deployed under <code>\site\wwwroot\app_data\Jobs\Triggered</code> .

### NOTE

A web app can time out after 20 minutes of inactivity, and only requests to the actual web app can reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site ([https://<app\\_name>.scm.azurewebsites.net](https://<app_name>.scm.azurewebsites.net)) doesn't reset the timer. If you set the web app that hosts your job to run continuously, run on a schedule, or use event-driven triggers, enable the **Always on** setting on your web app's Azure Configuration page. The Always on setting helps to make sure that these kinds of WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium pricing tiers.

## Supported file types for scripts or programs

The following file types are supported:

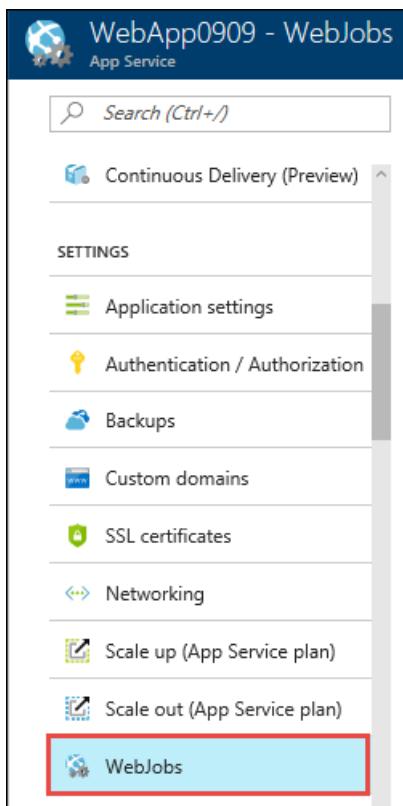
- .cmd, .bat, .exe (using Windows cmd)
- .ps1 (using PowerShell)
- .sh (using Bash)
- .php (using PHP)
- .py (using Python)
- .js (using Node.js)
- .jar (using Java)

## Create a continuous WebJob

### IMPORTANT

When you have source control configured for your application, WebJobs should be deployed as part of the source control integration. After source control is configured for your application, a WebJob can't be added from the Azure portal.

1. In the [Azure portal](#), go to the **App Service** page of your App Service web app, API app, or mobile app.
2. In the left pane of your app's **App Service** page, search for and select **WebJobs**.



3. On the **WebJobs** page, select **Add**.

The screenshot shows the Azure portal's 'WebJobs' blade. On the left, there's a sidebar with icons for 'WebJobs', 'Push', 'MySQL In App', 'Properties', 'Locks', and 'Automation script'. The main area has a title 'WebJobs' with a gear icon. Below it, a message says 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' A table with columns 'NAME', 'TYPE', 'STATUS', and 'SCHEDULE' is shown, with a note: 'You haven't added any WebJobs. Click ADD to get started.' At the top, there are buttons for '+ Add', 'Refresh', 'Logs', 'Delete', and 'Properties'.

4. Fill in the Add WebJob settings as specified in the table.

The dialog box is titled 'Add WebJob' and is associated with the app 'webapp0909'. It contains the following fields:

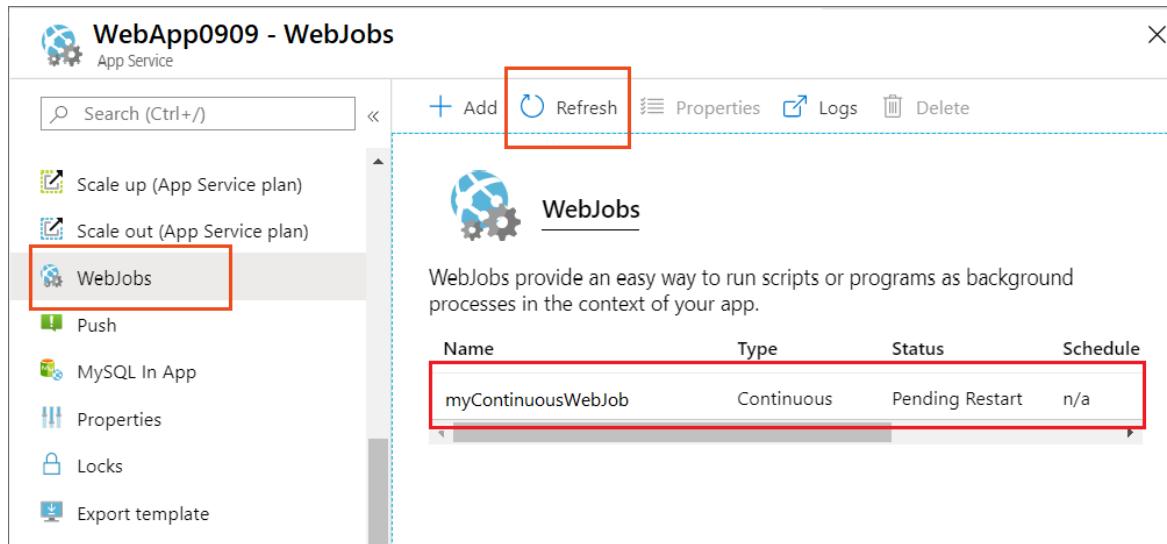
- Name**: myContinuousWebJob
- File Upload**: "ConsoleApp1.zip"
- Type**: Continuous
- Scale**: Multi Instance

At the bottom is a blue 'OK' button.

SETTING	SAMPLE VALUE	DESCRIPTION
Name	myContinuousWebJob	A name that is unique within an App Service app. Must start with a letter or a number and cannot contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A <i>.zip</i> file that contains your executable or script file as well as any supporting files needed to run the program or script. The supported executable or script file types are listed in the <a href="#">Supported file types</a> section.
Type	Continuous	The <a href="#">WebJob types</a> are described earlier in this article.
Scale	Multi instance	Available only for Continuous WebJobs. Determines whether the program or script runs on all instances or just one instance. The option to run on multiple instances doesn't apply to the Free or Shared <a href="#">pricing tiers</a> .

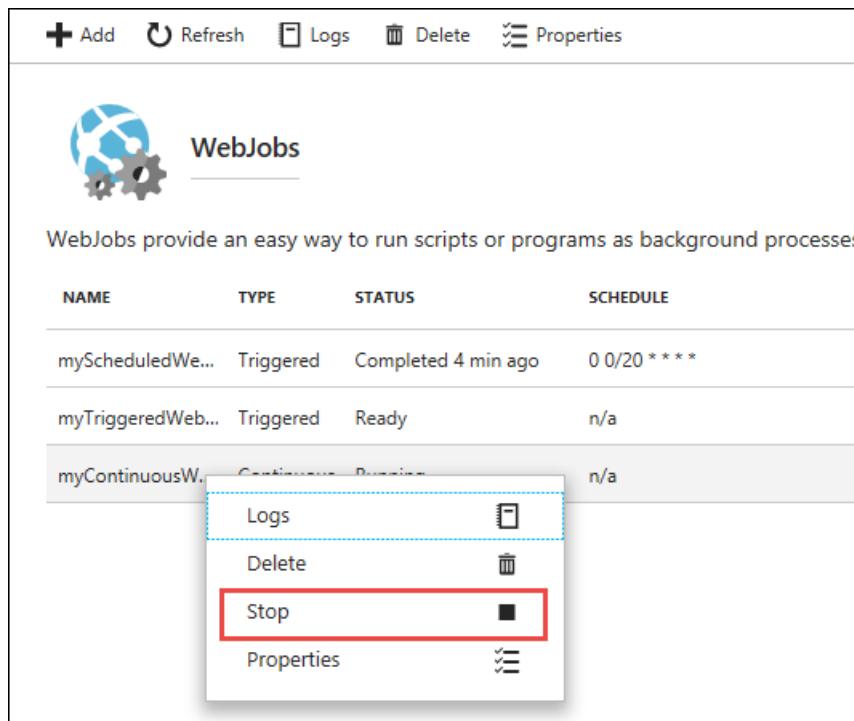
5. Select OK.

The new WebJob appears on the WebJobs page. If you see a message that says the WebJob was added, but you don't see it, select Refresh.



The screenshot shows the Azure portal's 'WebJobs' page for an app service named 'WebApp0909'. On the left, there's a sidebar with options like 'Scale up (App Service plan)', 'Scale out (App Service plan)', 'WebJobs' (which is selected and highlighted with a red box), 'Push', 'MySQL In App', 'Properties', 'Locks', and 'Export template'. The main area has a title 'WebJobs' with a sub-instruction: 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' Below this is a table with columns 'Name', 'Type', 'Status', and 'Schedule'. A single row is visible, showing 'myContinuousWebJob' as a 'Continuous' type job with 'Pending Restart' status and 'n/a' for the schedule. The entire row is also highlighted with a red box.

6. To stop or restart a continuous WebJob, right-click the WebJob in the list and select Stop or Start.



This screenshot shows the same 'WebJobs' page as the previous one, but with a context menu open over the 'myContinuousWebJob' row. The menu items are 'Logs' (highlighted with a blue box), 'Delete', 'Stop' (highlighted with a red box), and 'Properties'. The rest of the page remains the same, showing the list of WebJobs and their details.

## Create a manually triggered WebJob

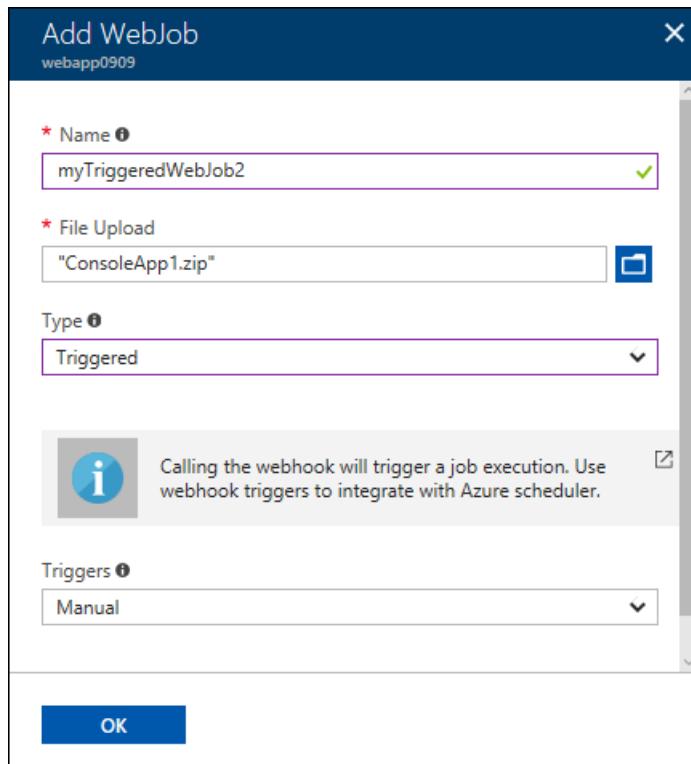
1. In the [Azure portal](#), search for and select **App Services**.
2. Select your web app, API app, or mobile app from the list.
3. In the left pane of your app's **App Service** page, select **WebJobs**.

The screenshot shows the Azure portal interface for a web application named 'WebApp0909'. The left sidebar lists various configuration options under 'SETTINGS': Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' option is highlighted with a red box.

4. On the **WebJobs** page, select **Add**.

The screenshot shows the 'WebJobs' page for the 'WebApp0909' application. The left sidebar shows 'WebJobs' selected. The main area features a large 'WebJobs' icon with the text 'WebJobs' below it. A descriptive message states: 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' Below this is a table with columns: NAME, TYPE, STATUS, and SCHEDULE. A note at the bottom says: 'You haven't added any WebJobs. Click ADD to get started.' A red box highlights the '+ Add' button in the top navigation bar.

5. Fill in the **Add WebJob** settings as specified in the table.



SETTING	SAMPLE VALUE	DESCRIPTION
Name	myTriggeredWebJob	A name that is unique within an App Service app. Must start with a letter or a number and cannot contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A .zip file that contains your executable or script file as well as any supporting files needed to run the program or script. The supported executable or script file types are listed in the <a href="#">Supported file types</a> section.
Type	Triggered	The <a href="#">WebJob types</a> are described previously in this article.
Triggers	Manual	

6. Select **OK**.

The new WebJob appears on the **WebJobs** page. If you see a message that says the WebJob was added, but you don't see it, select **Refresh**.

7. To run the WebJob, right-click its name in the list and select Run.

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Ready	0 0/20 * * *
myTriggeredWeb...	Triggered	Ready	n/a
myContinuousW...	Continuous	Pending Restart	n/a

## Create a scheduled WebJob

A scheduled Webjob is also triggered. You can schedule the trigger to occur automatically on the schedule you specify.

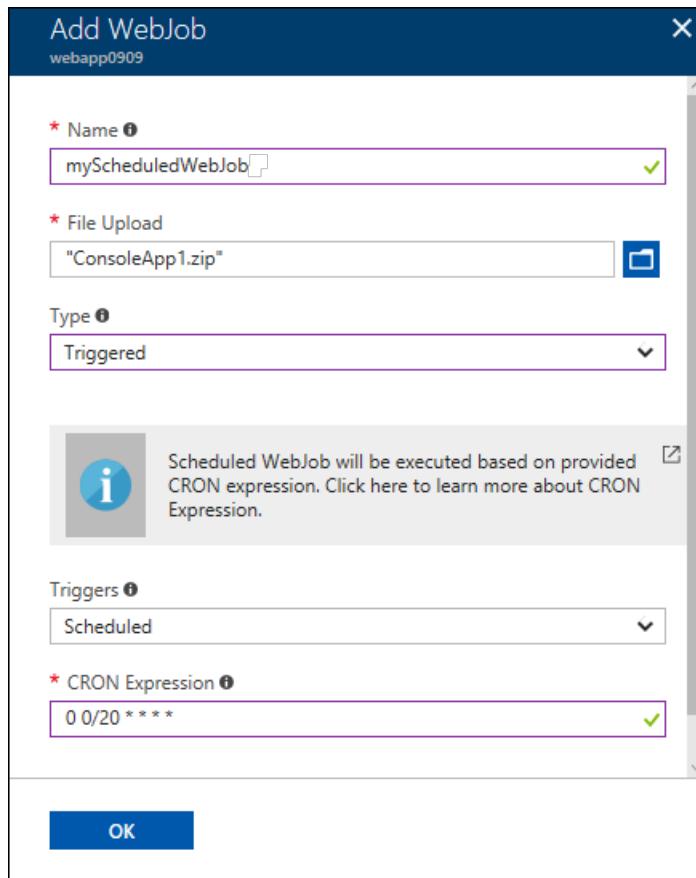
1. In the [Azure portal](#), search for and select **App Services**.
2. Select your web app, API app, or mobile app from the list.
3. In the left pane of your app's **App Service** page, select **WebJobs**.

The screenshot shows the Azure portal interface for a web application named 'WebApp0909'. The left sidebar lists various configuration options under 'SETTINGS', including Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, Scale up (App Service plan), Scale out (App Service plan), and WebJobs. The 'WebJobs' option is highlighted with a red box.

4. On the **WebJobs** page, select **Add**.

The screenshot shows the 'WebJobs' page for the 'WebApp0909' application. The left sidebar shows 'WebJobs' selected. The main area features a 'WebJobs' icon and the text: 'WebJobs provide an easy way to run scripts or programs as background processes in the context of your app.' A prominent red box highlights the '+ Add' button in the top navigation bar. Below it, there's a table header with columns: NAME, TYPE, STATUS, and SCHEDULE. A message at the bottom states: 'You haven't added any WebJobs. Click ADD to get started.'

5. Fill in the **Add WebJob** settings as specified in the table.



SETTING	SAMPLE VALUE	DESCRIPTION
Name	myScheduledWebJob	A name that is unique within an App Service app. Must start with a letter or a number and cannot contain special characters other than "-" and "_".
File Upload	ConsoleApp.zip	A .zip file that contains your executable or script file as well as any supporting files needed to run the program or script. The supported executable or script file types are listed in the <a href="#">Supported file types</a> section.
Type	Triggered	The <a href="#">WebJob types</a> are described earlier in this article.
Triggers	Scheduled	For the scheduling to work reliably, enable the Always On feature. Always On is available only in the Basic, Standard, and Premium pricing tiers.
CRON Expression	0 0/20 * * *	<a href="#">CRON expressions</a> are described in the following section.

## 6. Select OK.

The new WebJob appears on the [WebJobs](#) page. If you see a message that says the WebJob was added, but you don't see it, select **Refresh**.

Name	Type	Status	Schedule
myScheduledWebJob	Triggered	Ready	0 0/15 * * *
myTriggeredWebJob	Triggered	Ready	n/a
myContinuousWebJob	Continuous	Pending Restart	n/a

## NCRONTAB expressions

You can enter a **NCRONTAB expression** in the portal or include a `settings.job` file at the root of your WebJob `.zip` file, as in the following example:

```
{
 "schedule": "0 */15 * * *"
}
```

To learn more, see [Scheduling a triggered WebJob](#).

### NOTE

The default time zone used to run CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression run based on another time zone, create an app setting for your function app named `WEBSITE_TIME_ZONE`. To learn more, see [NCRONTAB time zones](#).

## Manage WebJobs

You can manage the running state individual WebJobs running in your site in the [Azure portal](#). Just go to **Settings > WebJobs**, choose the WebJob, and you can start and stop the WebJob. You can also view and modify the password of the webhook that runs the WebJob.

You can also [add an application setting](#) named `WEBJOB_STOPPED` with a value of `1` to stop all WebJobs running on your site. This can be handy as a way to prevent conflicting WebJobs from running both in staging and production slots. You can similarly use a value of `1` for the `WEBJOBS_DISABLE_SCHEDULE` setting to disable triggered WebJobs in the site or a staging slot. For slots, remember to enable the **Deployment slot setting** option so that the setting itself doesn't get swapped.

## View the job history

1. Select the WebJob and then to see the history, select **Logs**.

The screenshot shows the 'WebApp0909 - WebJobs' page in the Azure portal. On the left, there's a sidebar with 'SETTINGS' and various options like Application settings, Authentication / Authorization, Backups, Custom domains, SSL certificates, Networking, and Scale up (App Service plan). The main area has a title 'WebJobs' with a gear icon. Below it, a description says 'WebJobs provide an easy way to run scripts or programs as background processes'. A table lists three jobs:

NAME	TYPE	STATUS	SCHEDULE
myScheduledWe...	Triggered	Completed 5 min ago	0 0/20 * * *
myTriggeredWeb...	Triggered	Completed Just now	n/a
myContinuousW...	Continuous	Running	n/a

2. In the **WebJob Details** page, select a time to see details for one run.

The screenshot shows the 'Microsoft Azure WebJobs' page for 'myTriggeredWebJob'. At the top, it says 'Run command: ConsoleApp1.exe'. Below that, a section titled 'Recent job runs' shows two entries:

TIMING	STATUS
13 minutes ago (203 ms running time)	Success
16 minutes ago (361 ms running time)	Success

At the bottom, there's a note: 'Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables.'

3. In the **WebJob Run Details** page, select **Toggle Output** to see the text of the log contents.

**Microsoft Azure WebJobs**

WebJobs / myTriggeredWebJob

## WebJob Run Details myTriggeredWebJob

**Success** 20 minutes ago (203 ms running time)  
Run ID: 201709112047171935

**Toggle Output**

download

```
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Status changed to Initializing
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Run script 'ConsoleApp1.exe' with script host - 'WindowsScriptHost'
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Status changed to Running
[09/11/2017 20:47:17 > 9ed5b3: INFO] Hello World!
[09/11/2017 20:47:17 > 9ed5b3: SYS INFO] Status changed to Success
```

Do more with [Microsoft Azure WebJobs SDK](#). The SDK integrates Microsoft Azure Storage, triggering a function in your program when items are added to Queues, Blobs, or Tables.

To see the output text in a separate browser window, select **download**. To download the text itself, right-click **download** and use your browser options to save the file contents.

4. Select the **WebJobs** breadcrumb link at the top of the page to go to a list of WebJobs.

**Microsoft Azure WebJobs**

**WebJobs** / myTriggeredWebJob

← → ⏪ ites.net/azurejobs/#/jobs 📄 ⭐ | ⚡ ⚡ ⌂ ...

## Microsoft Azure WebJobs

### WebJobs

NAME	STATUS	LAST RUN TIME
myScheduledWebJob	Success	14 minutes ago (328 ms)
myTriggeredWebJob	Success	27 minutes ago (203 ms)
myContinuousWebJob	Running	Runs continuously

## Next steps

The Azure WebJobs SDK can be used with WebJobs to simplify many programming tasks. For more

information, see [What is the WebJobs SDK](#).

# Develop and deploy WebJobs using Visual Studio

11/2/2021 • 10 minutes to read • [Edit Online](#)

This article explains how to use Visual Studio to deploy a console app project to a web app in [Azure App Service](#) as an [Azure WebJob](#). For information about how to deploy WebJobs by using the [Azure portal](#), see [Run background tasks with WebJobs in Azure App Service](#).

You can choose to develop a WebJob that runs as either a [.NET Core app](#) or a [.NET Framework app](#). Version 3.x of the [Azure WebJobs SDK](#) lets you develop WebJobs that run as either .NET Core apps or .NET Framework apps, while version 2.x supports only the .NET Framework. The way that you deploy a WebJobs project is different for .NET Core projects than for .NET Framework projects.

You can publish multiple WebJobs to a single web app, provided that each WebJob in a web app has a unique name.

## WebJobs as .NET Core console apps

With version 3.x of the Azure WebJobs SDK, you can create and publish WebJobs as .NET Core console apps. For step-by-step instructions to create and publish a .NET Core console app to Azure as a WebJob, see [Get started with the Azure WebJobs SDK for event-driven background processing](#).

### NOTE

.NET Core Web Apps and/or .NET Core WebJobs can't be linked with web projects. If you need to deploy your WebJob with a web app, [create your WebJobs as a .NET Framework console app](#).

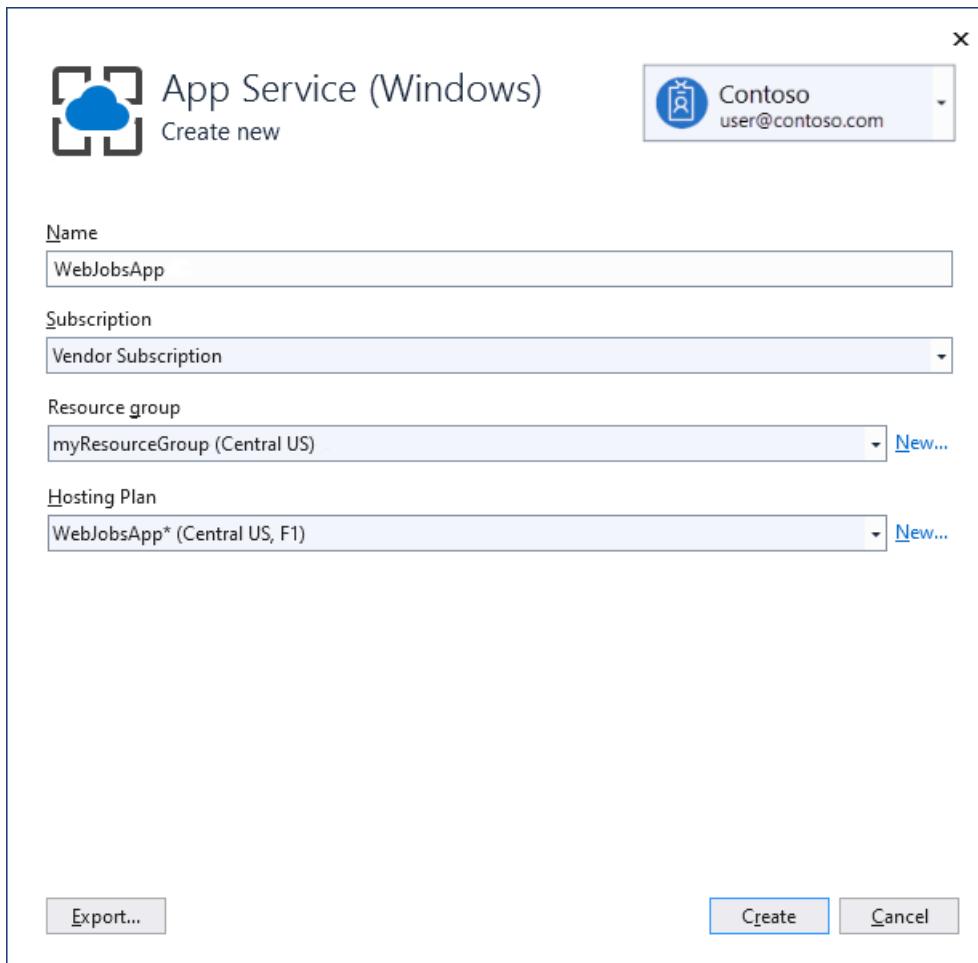
### Deploy to Azure App Service

Publishing a .NET Core WebJob to Azure App Service from Visual Studio uses the same tooling as publishing an ASP.NET Core app.

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog box, select **Azure** for **Target**, and then select **Next**.
3. Select **Azure WebJobs** for **Specific target**, and then select **Next**.
4. Above **App Service instances** select the plus (+) button to **Create a new Azure WebJob**.
5. In the **App Service (Windows)** dialog box, use the hosting settings in the following table.

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource group	myResourceGroup	Name of the resource group in which to create your function app. Choose <b>New</b> to create a new resource group.

SETTING	SUGGESTED VALUE	DESCRIPTION
Hosting Plan	App Service plan	An <a href="#">App Service plan</a> specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan. App Service plans define the region, instance size, scale count, and SKU (Free, Shared, Basic, Standard, or Premium). Choose <b>New</b> to create a new App Service plan. Free and Basic tiers don't support the Always On option to keep your site running continuously.



6. Select **Create** to create a WebJob and related resources in Azure with these settings and deploy your project code.
7. Select **Finish** to return to the **Publish** page.

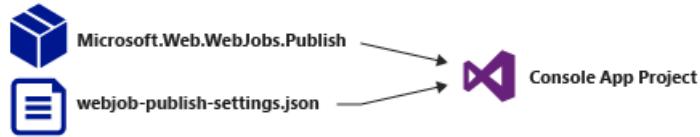
## WebJobs as .NET Framework console apps

If you use Visual Studio to deploy a WebJobs-enabled .NET Framework console app project, it copies runtime files to the appropriate folder in the web app (*App\_Data/jobs/continuous* for continuous WebJobs and *App\_Data/jobs/triggered* for scheduled or on-demand WebJobs).

Visual Studio adds the following items to a WebJobs-enabled project:

- The [Microsoft.Web.WebJobs.Publish](#) NuGet package.
- A [webjob-publish-settings.json](#) file that contains deployment and scheduler settings.

## Enable deployment as a WebJob



You can add these items to an existing console app project or use a template to create a new WebJobs-enabled console app project.

Deploy a project as a WebJob by itself, or link it to a web project so that it automatically deploys whenever you deploy the web project. To link projects, Visual Studio includes the name of the WebJobs-enabled project in a [webjobs-list.json](#) file in the web project.

## Link WebJob project to web project



## Prerequisites

Install Visual Studio 2017 or Visual Studio 2019 with the [Azure development workload](#).

### Enable WebJobs deployment for an existing console app project

You have two options:

- [Enable automatic deployment with a web project](#).

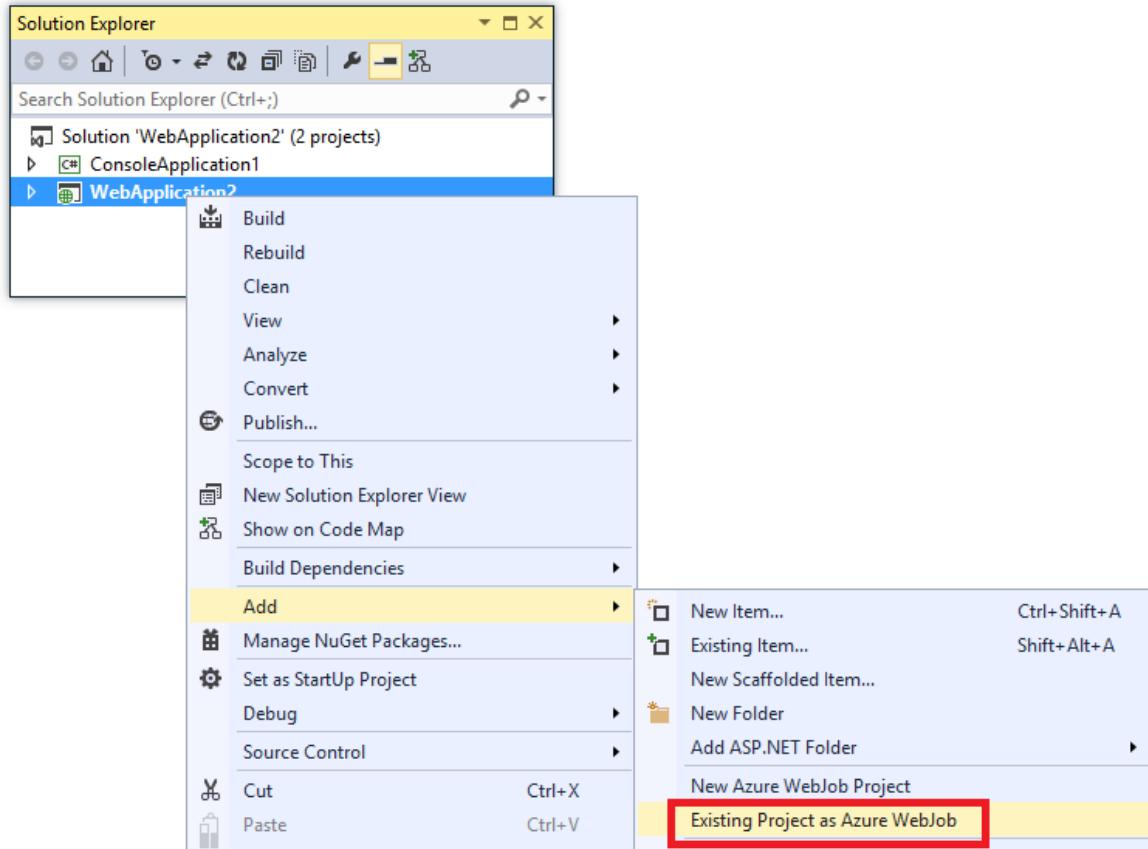
Configure an existing console app project so that it automatically deploys as a WebJob when you deploy a web project. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

- [Enable deployment without a web project](#).

Configure an existing console app project to deploy as a WebJob by itself, without a link to a web project. Use this option when you want to run a WebJob in a web app by itself, with no web application running in the web app. You might want to do so to scale your WebJob resources independently of your web application resources.

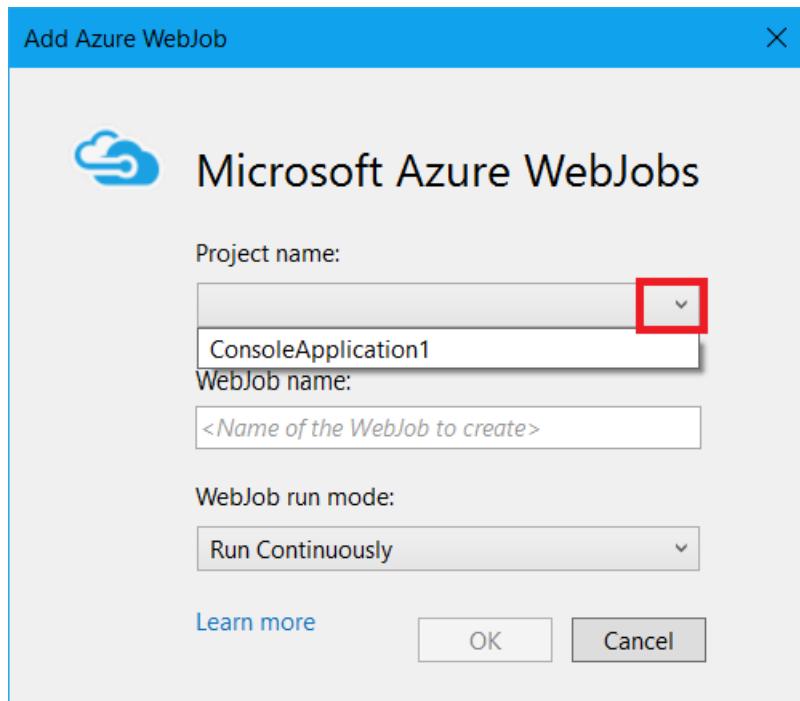
### Enable automatic WebJobs deployment with a web project

1. Right-click the web project in **Solution Explorer**, and then select **Add > Existing Project as Azure WebJob**.



The [Add Azure WebJob](#) dialog box appears.

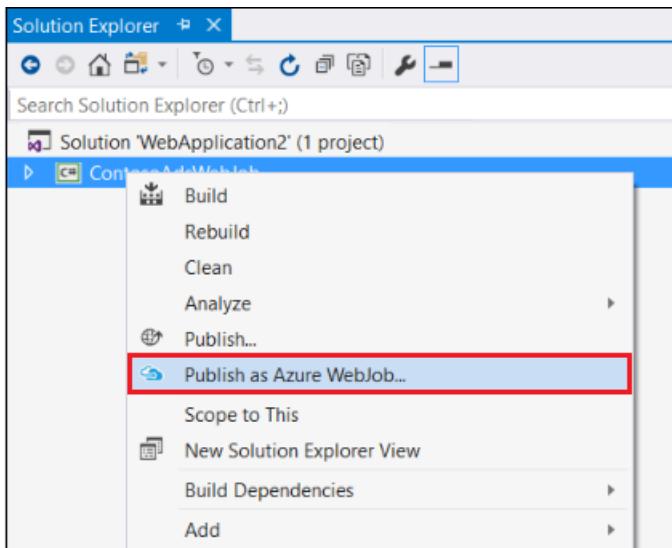
2. In the **Project name** drop-down list, select the console app project to add as a WebJob.



3. Complete the [Add Azure WebJob](#) dialog box, and then select **OK**.

#### Enable WebJobs deployment without a web project

1. Right-click the console app project in **Solution Explorer**, and then select **Publish as Azure WebJob**.



The [Add Azure WebJob](#) dialog box appears, with the project selected in the **Project name** box.

2. Complete the [Add Azure WebJob](#) dialog box, and then select **OK**.

The **Publish Web** wizard appears. If you don't want to publish immediately, close the wizard. The settings that you've entered are saved for when you do want to [deploy the project](#).

### Create a new WebJobs-enabled project

To create a new WebJobs-enabled project, use the console app project template and enable WebJobs deployment as explained in [the previous section](#). As an alternative, you can use the WebJobs new-project template:

- [Use the WebJobs new-project template for an independent WebJob](#)

Create a project and configure it to deploy by itself as a WebJob, with no link to a web project. Use this option when you want to run a WebJob in a web app by itself, with no web application running in the web app. You might want to do so to scale your WebJob resources independently of your web application resources.

- [Use the WebJobs new-project template for a WebJob linked to a web project](#)

Create a project that is configured to deploy automatically as a WebJob when you deploy a web project in the same solution. Use this option when you want to run your WebJob in the same web app in which you run the related web application.

#### NOTE

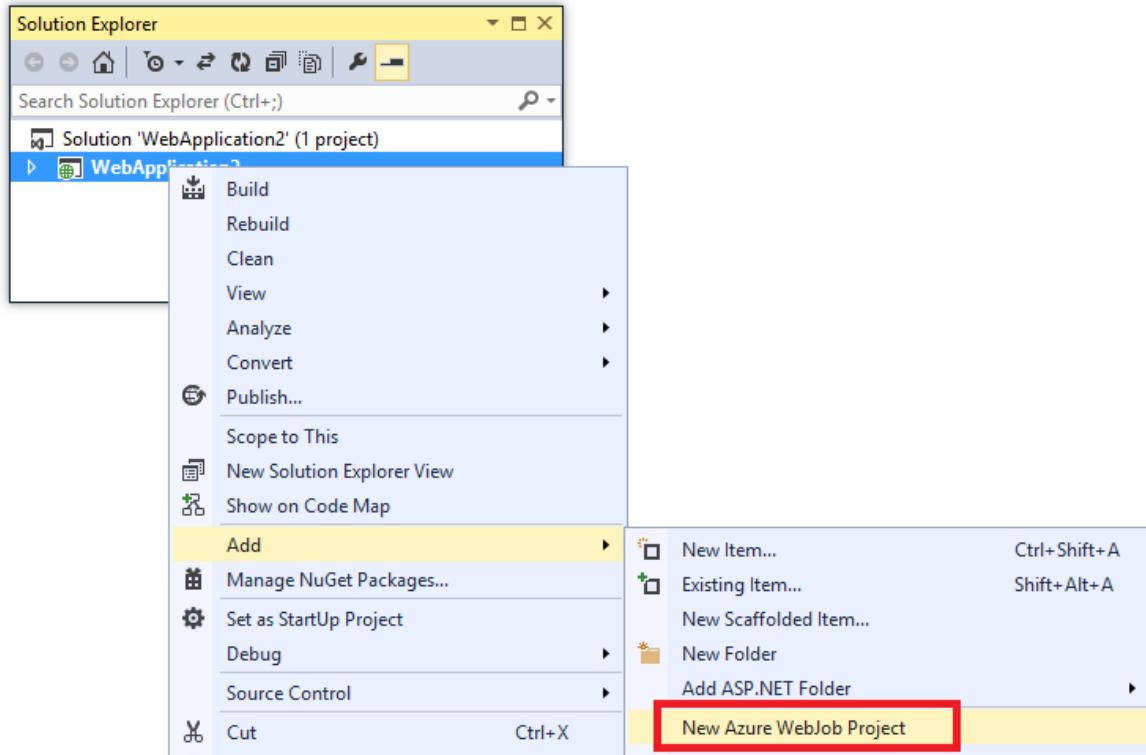
The WebJobs new-project template automatically installs NuGet packages and includes code in *Program.cs* for the [WebJobs SDK](#). If you don't want to use the WebJobs SDK, remove or change the `host.RunAndBlock` statement in *Program.cs*.

#### Use the WebJobs new-project template for an independent WebJob

1. Select **File > New > Project**. In the [Create a new project](#) dialog box, search for and select **Azure WebJob (.NET Framework)** for C#.
2. Follow the previous directions to [make the console app project an independent WebJobs project](#).

#### Use the WebJobs new-project template for a WebJob linked to a web project

1. Right-click the web project in **Solution Explorer**, and then select **Add > New Azure WebJob Project**.



The [Add Azure WebJob](#) dialog box appears.

2. Complete the [Add Azure WebJob](#) dialog box, and then select **OK**.

#### **webjob-publish-settings.json** file

When you configure a console app for WebJobs deployment, Visual Studio installs the [Microsoft.Web.WebJobs.Publish](#) NuGet package and stores scheduling information in a *webjob-publish-settings.json* file in the project *Properties* folder of the WebJobs project. Here is an example of that file:

```
{
 "$schema": "http://schemastore.org/schemas/json/webjob-publish-settings.json",
 "webJobName": "WebJob1",
 "startTime": "null",
 "endTime": "null",
 "jobRecurrenceFrequency": "null",
 "interval": null,
 "runMode": "Continuous"
}
```

You can edit this file directly, and Visual Studio provides IntelliSense. The file schema is stored at <https://schemastore.org> and can be viewed there.

#### **webjobs-list.json** file

When you link a WebJobs-enabled project to a web project, Visual Studio stores the name of the WebJobs project in a *webjobs-list.json* file in the web project's *Properties* folder. The list might contain multiple WebJobs projects, as shown in the following example:

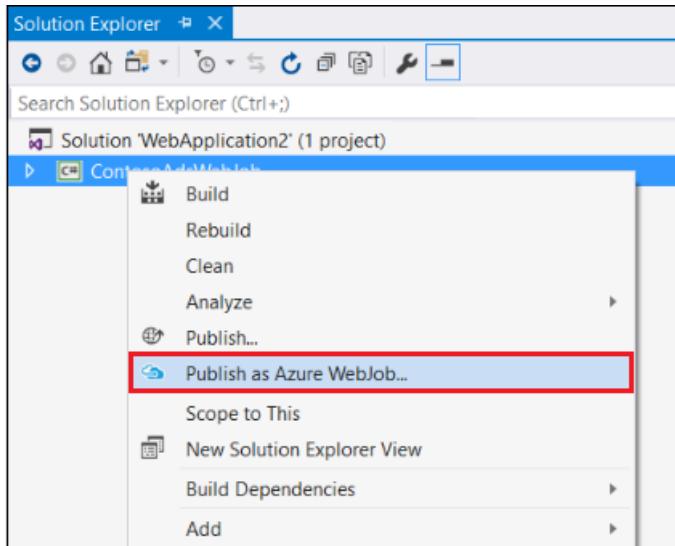
```
{
 "$schema": "http://schemastore.org/schemas/json/webjobs-list.json",
 "WebJobs": [
 {
 "filePath": "../ConsoleApplication1/ConsoleApplication1.csproj"
 },
 {
 "filePath": "../WebJob1/WebJob1.csproj"
 }
]
}
```

You can edit this file directly in Visual Studio, with IntelliSense. The file schema is stored at <https://schemastore.org>.

### Deploy a WebJobs project

A WebJobs project that you've linked to a web project deploys automatically with the web project. For information about web project deployment, see [How-to guides > Deploy the app](#) in the left navigation.

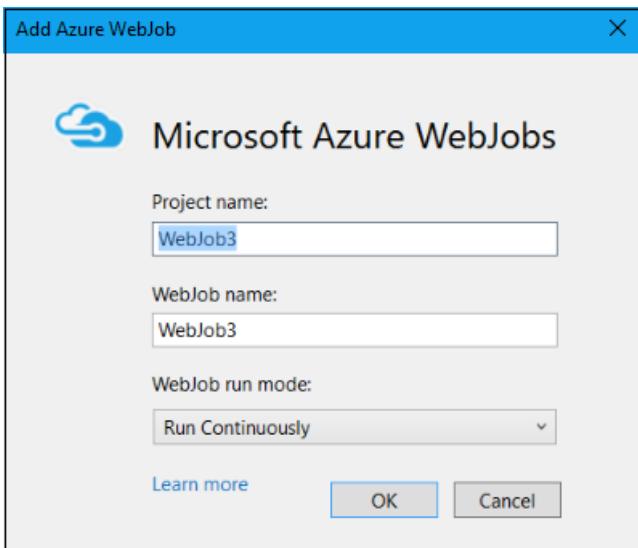
To deploy a WebJobs project by itself, right-click the project in **Solution Explorer** and select **Publish as Azure WebJob**.



For an independent WebJob, the same **Publish Web** wizard that is used for web projects appears, but with fewer settings available to change.

### Add Azure WebJob dialog box

The **Add Azure WebJob** dialog box lets you enter the WebJob name and the run mode setting for your WebJob.



Some of the fields in this dialog box correspond to fields on the **Add WebJob** dialog box of the Azure portal. For more information, see [Run background tasks with WebJobs in Azure App Service](#).

WebJob deployment information:

- For information about command-line deployment, see [Enabling Command-line or Continuous Delivery of Azure WebJobs](#).
- If you deploy a WebJob, and then decide you want to change the type of WebJob and redeploy, delete the `webjobs-publish-settings.json` file. Doing so causes Visual Studio to redisplay the publishing options, so you can change the type of WebJob.
- If you deploy a WebJob and later change the run mode from continuous to non-continuous or vice versa, Visual Studio creates a new WebJob in Azure when you redeploy. If you change other scheduling settings, but leave run mode the same or switch between Scheduled and On Demand, Visual Studio updates the existing job instead of creating a new one.

## WebJob types

The type of a WebJob can be either *triggered* or *continuous*:

- Triggered (default): A triggered WebJob starts based on a binding event, on a `schedule`, or when you trigger it manually (on demand). It runs on a single instance that the web app runs on.
- Continuous: A `continuous` WebJob starts immediately when the WebJob is created. It runs on all web app scaled instances by default but can be configured to run as a single instance via `settings.job`.

### NOTE

A web app can time out after 20 minutes of inactivity, and only requests to the actual web app can reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (`https://<app_name>.scm.azurewebsites.net`) doesn't reset the timer. If you set the web app that hosts your job to run continuously, run on a schedule, or use event-driven triggers, enable the **Always on** setting on your web app's Azure Configuration page. The Always on setting helps to make sure that these kinds of WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium pricing tiers.

### Scheduling a triggered WebJob

When you publish a console app to Azure, Visual Studio sets the type of WebJob to **Triggered** by default, and adds a new `settings.job` file to the project. For triggered WebJob types, you can use this file to set an execution schedule for your WebJob.

Use the `settings.job` file to set an execution schedule for your WebJob. The following example runs every hour from 9 AM to 5 PM:

```
{
 "schedule": "0 0 9-17 * * *"
}
```

This file is located at the root of the WebJobs folder with your WebJob's script, such as

`wwwroot\app_data\jobs\triggered\{job name}` or `wwwroot\app_data\jobs\continuous\{job name}`. When you deploy a WebJob from Visual Studio, mark your `settings.job` file properties in Visual Studio as **Copy if newer**.

If you [create a WebJob from the Azure portal](#), the `settings.job` file is created for you.

#### CRON expressions

WebJobs uses the same CRON expressions for scheduling as the timer trigger in Azure Functions. To learn more about CRON support, see [Timer trigger for Azure Functions](#).

#### NOTE

The default time zone used to run CRON expressions is Coordinated Universal Time (UTC). To have your CRON expression run based on another time zone, create an app setting for your function app named `WEBSITE_TIME_ZONE`. To learn more, see [NCRONTAB time zones](#).

#### settings.job reference

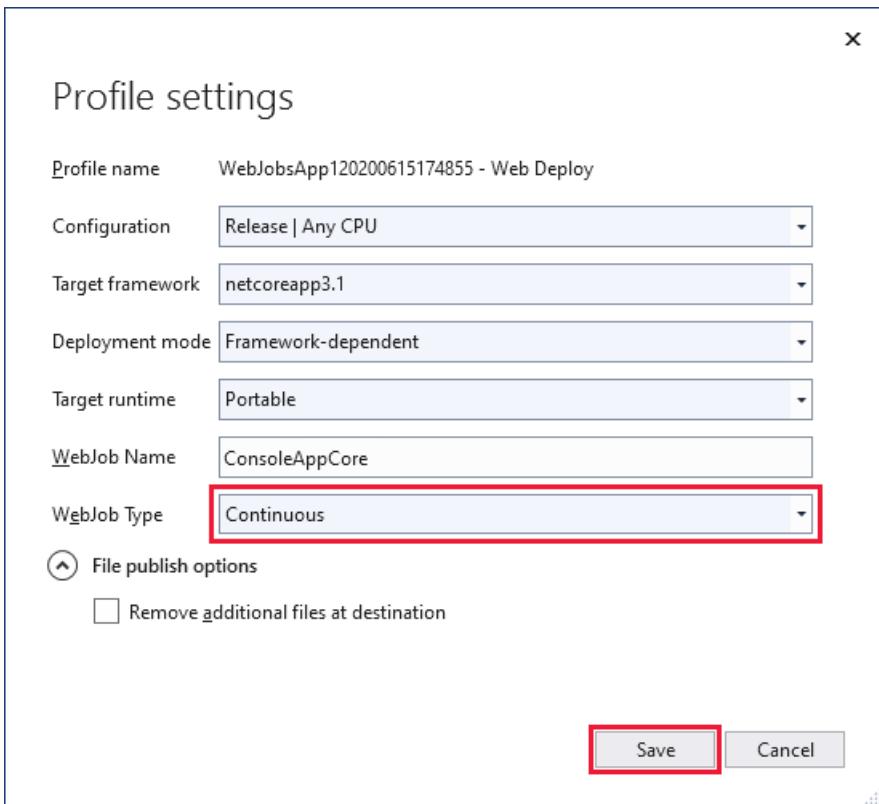
The following settings are supported by WebJobs:

SETTING	TYPE	DESCRIPTION
<code>is_in_place</code>	All	Allows the WebJob to run in place without first being copied to a temporary folder. For more information, see <a href="#">WebJob working directory</a> .
<code>is_singleton</code>	Continuous	Only run the WebJob on a single instance when scaled out. For more information, see <a href="#">Set a continuous job as singleton</a> .
<code>schedule</code>	Triggered	Run the WebJob on a CRON-based schedule. For more information, see <a href="#">NCRONTAB expressions</a> .
<code>stopping_wait_time</code>	All	Allows control of the shutdown behavior. For more information, see <a href="#">Graceful shutdown</a> .

#### Continuous execution

If you enable **Always on** in Azure, you can use Visual Studio to change the WebJob to run continuously:

1. If you haven't already done so, [publish the project to Azure](#).
2. In **Solution Explorer**, right-click the project and select **Publish**.
3. In the **Publish** tab, choose **Edit**.
4. In the **Profile settings** dialog box, choose **Continuous** for **WebJob Type**, and then choose **Save**.



5. Select **Publish** in the **Publish** tab to republish the WebJob with the updated settings.

## Next steps

[Learn more about the WebJobs SDK](#)

# Tutorial: Get started with the Azure WebJobs SDK for event-driven background processing

11/2/2021 • 16 minutes to read • [Edit Online](#)

Get started with the Azure WebJobs SDK for Azure App Service to enable your web apps to run background tasks, scheduled tasks, and respond to events.

Use Visual Studio 2019 to create a .NET core console app that uses the WebJobs SDK to respond to Azure Storage Queue messages, run the project locally, and finally deploy it to Azure.

In this tutorial, you will learn how to:

- Create a console app
- Add a function
- Test locally
- Deploy to Azure
- Enable Application Insights logging
- Add input/output bindings

## Prerequisites

- Visual Studio 2019 with the **Azure development** workload. [Install Visual Studio 2019](#).
- An Azure account with an active subscription. [Create an account for free](#).

## Create a console app

In this section, you start by creating a project in Visual Studio 2019. Next, you'll add tools for Azure development, code publishing, and functions that listen for triggers and call functions. Last, you'll set up console logging that disables a legacy monitoring tool and enables a console provider with default filtering.

### NOTE

The procedures in this article are verified for creating a .NET Core console app that runs on .NET Core 3.1.

### Create a project

1. In Visual Studio, select **File > New > Project**.
2. Under **Create a new project**, select **Console Application (C#)**, and then select **Next**.
3. Under **Configure your new project**, name the project *WebJobsSDKSample*, and then select **Next**.
4. Choose your **Target framework** and select **Create**. This tutorial has been verified using .NET Core 3.1.

### Install WebJobs NuGet packages

Install the latest WebJobs NuGet package. This package includes Microsoft.Azure.WebJobs (WebJobs SDK), which lets you publish your function code to WebJobs in Azure App Service.

1. Get the latest stable 3.x version of the [Microsoft.Azure.WebJobs.Extensions NuGet package](#).
2. In Visual Studio, go to **Tools > NuGet Package Manager**.

3. Select **Package Manager Console**. You'll see a list of NuGet cmdlets, a link to documentation, and a **PM>** entry point.

4. In the following command, replace **<3\_X\_VERSION>** with the current version number you found in step 1.

```
Install-Package Microsoft.Azure.WebJobs.Extensions -version <3_X_VERSION>
```

5. In the **Package Manager Console**, execute the command. The extension list appears and automatically installs.

## Create the Host

The host is the runtime container for functions that listens for triggers and calls functions. The following steps create a host that implements **IHost**, which is the Generic Host in ASP.NET Core.

1. Select the **Program.cs** tab and add these **using** statements:

```
using System.Threading.Tasks;
using Microsoft.Extensions.Hosting;
```

2. Also under **Program.cs**, replace the **Main** method with the following code:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

In ASP.NET Core, host configurations are set by calling methods on the **HostBuilder** instance. For more information, see [.NET Generic Host](#). The **ConfigureWebJobs** extension method initializes the WebJobs host. In **ConfigureWebJobs**, initialize specific binding extensions, such as the Storage binding extension, and set properties of those extensions.

## Enable console logging

Set up console logging that uses the [ASP.NET Core logging framework](#). This framework, `Microsoft.Extensions.Logging`, includes an API that works with a variety of built-in and third-party logging providers.

1. Get the latest stable version of the [Microsoft.Extensions.Logging.Console](#) NuGet package, which includes `Microsoft.Extensions.Logging`.

2. In the following command, replace **<3\_X\_VERSION>** with the current version number you found in step 1. Each type of NuGet Package has a unique version number.

```
Install-Package Microsoft.Extensions.Logging.Console -version <3_X_VERSION>
```

3. In the **Package Manager Console**, fill in the current version number and execute the command. The extension list appears and automatically installs.

4. Under the tab **Program.cs**, add this `using` statement:

```
using Microsoft.Extensions.Logging;
```

5. Continuing under **Program.cs**, add the `ConfigureLogging` method to `HostBuilder`, before the `Build` command. The `AddConsole` method adds console logging to the configuration.

```
builder.ConfigureLogging((context, b) =>
{
 b.AddConsole();
});
```

The `Main` method now looks like this:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 });
 builder.ConfigureLogging((context, b) =>
 {
 b.AddConsole();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

This addition makes these changes:

- Disables [dashboard logging](#). The dashboard is a legacy monitoring tool, and dashboard logging is not recommended for high-throughput production scenarios.
- Adds the console provider with default [filtering](#).

Now, you can add a function that is triggered by messages arriving in an Azure Storage queue.

## Add a function

A function is unit of code that runs on a schedule, is triggered based on events, or is run on demand. A trigger listens to a service event. In the context of the WebJobs SDK, triggered doesn't refer to the deployment mode. Event-driven or scheduled WebJobs created using the SDK should always be deployed as continuous WebJobs with "Always on" enabled.

In this section, you create a function triggered by messages in an Azure Storage queue. First, you need to add a binding extension to connect to Azure Storage.

### Install the Storage binding extension

Starting with version 3 of the WebJobs SDK, to connect to Azure Storage services you must install a separate Storage binding extension package.

1. Get the latest stable version of the [Microsoft.Azure.WebJobs.Extensions.Storage](#) NuGet package, version 3.x.

2. In the following command, replace `<3_X_VERSION>` with the current version number you found in step 1.

Each type of NuGet Package has a unique version number.

```
Install-Package Microsoft.Azure.WebJobs.Extensions.Storage -Version <3_X_VERSION>
```

3. In the **Package Manager Console**, execute the command with the current version number at the `PM>` entry point.

4. Continuing in `Program.cs`, in the `ConfigureWebJobs` extension method, add the `AddAzureStorage` method on the `HostBuilder` instance (before the `Build` command) to initialize the Storage extension. At this point, the `ConfigureWebJobs` method looks like this:

```
builder.ConfigureWebJobs(b =>
{
 b.AddAzureStorageCoreServices();
 b.AddAzureStorage();
});
```

5. Add the following code in the `Main` method after the `builder` is instantiated:

```
builder.UseEnvironment(EnvironmentName.Development);
```

Running in [development mode](#) reduces the [queue polling exponential backoff](#) that can significantly delay the amount of time it takes for the runtime to find the message and invoke the function. You should remove this line of code or switch to `Production` when you're done with development and testing.

The `Main` method should now look like the following example:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.UseEnvironment(EnvironmentName.Development);
 builder.ConfigureLogging((context, b) =>
 {
 b.AddConsole();
 });
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddAzureStorage();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

## Create a queue triggered function

The `queueTrigger` attribute tells the runtime to call this function when a new message is written on an Azure Storage queue called `queue`. The contents of the queue message are provided to the method code in the `message` parameter. The body of the method is where you process the trigger data. In this example, the code just logs the message.

1. In Solution Explorer, right-click the project, select **Add > New Item**, and then select **Class**.

2. Name the new C# class file `Functions.cs` and select **Add**.

3. In *Functions.cs*, replace the generated template with the following code:

```
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;

namespace WebJobsSDKSample
{
 public class Functions
 {
 public static void ProcessQueueMessage([QueueTrigger("queue")] string message, ILogger
logger)
 {
 logger.LogInformation(message);
 }
 }
}
```

You should mark the *Functions* class as `public static` in order for the runtime to access and execute the method. In the above code sample, when a message is added to a queue named `queue`, the function executes and the `message` string is written to the logs. The queue being monitored is in the default Azure Storage account, which you create next.

The `message` parameter doesn't have to be a string. You can also bind to a JSON object, a byte array, or a [CloudQueueMessage](#) object. [See Queue trigger usage](#). Each binding type (such as queues, blobs, or tables) has a different set of parameter types that you can bind to.

### Create an Azure storage account

The Azure Storage Emulator that runs locally doesn't have all of the features that the WebJobs SDK needs. You'll create a storage account in Azure and configure the project to use it.

To learn how to create a general-purpose v2 storage account, see [Create an Azure Storage account](#).

### Locate and copy your connection string

A connection string is required to configure storage. Keep this connection string for the next steps.

1. In the [Azure portal](#), navigate to your storage account and select **Settings**.
2. In **Settings**, select **Access keys**.
3. For the **Connection string** under **key1**, select the **Copy to clipboard** icon.

The screenshot shows the 'Access keys' section of the Azure Storage account settings. The left sidebar has a red box around the 'Access keys' item. The main area contains a note about using access keys for authentication, a 'Storage account name' input field with 'cs41003000ad2d7e07', and a 'key1' entry with a red box around it. Below it is a 'Connection string' input field.

## Configure storage to run locally

The WebJobs SDK looks for the storage connection string in the Application Settings in Azure. When you run locally, it looks for this value in the local configuration file or in environment variables.

1. Right-click the project, select **Add > New Item**, select **JavaScript JSON configuration file**, name the new file **appsettings.json**, and select **Add**.
2. In the new file, add a **AzureWebJobsStorage** field, as in the following example:

```
{
 "AzureWebJobsStorage": "{storage connection string}"
}
```

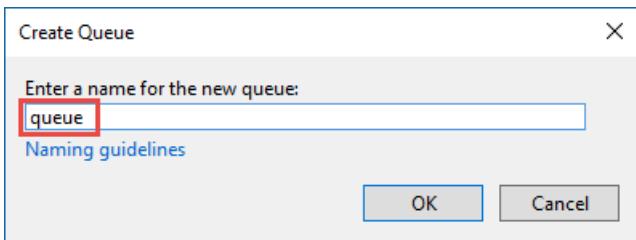
3. Replace *{storage connection string}* with the connection string that you copied previously.
4. Select the **appsettings.json** file in Solution Explorer and in the **Properties** window, set the **Copy to Output Directory** action to **Copy if newer**.

Because this file contains a connection string secret, you shouldn't store the file in a remote code repository. After publishing your project to Azure, you can add the same connection string app setting in your app in Azure App Service.

## Test locally

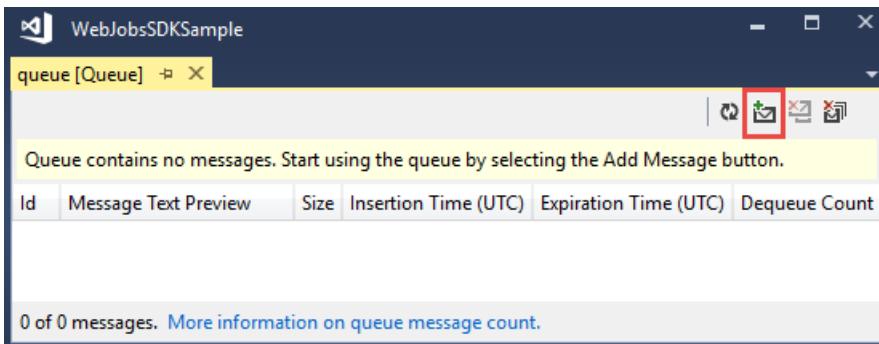
Build and run the project locally and create a message queue to trigger the function.

1. In **Cloud Explorer** in Visual Studio, expand the node for your new storage account, and then right-click **Queues**.
2. Select **Create Queue**.
3. Enter *queue* as the name for the queue, and then select **OK**.

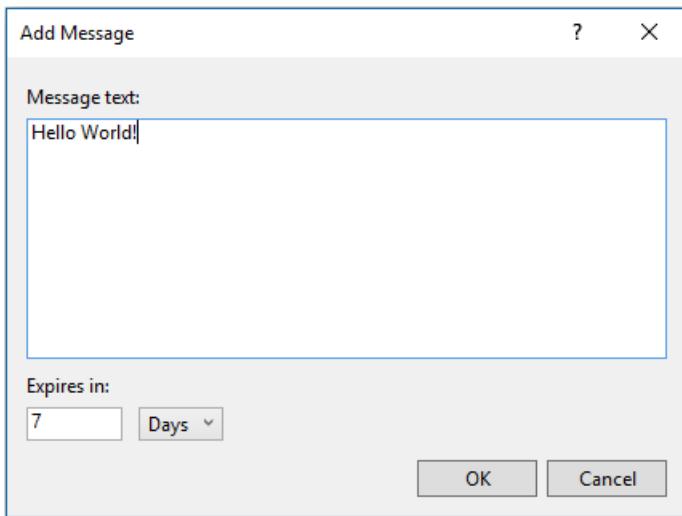


4. Right-click the node for the new queue, and then select **Open**.

5. Select the **Add Message** icon.



6. In the **Add Message** dialog, enter *Hello World!* as the **Message text**, and then select **OK**. There is now a message in the queue.



7. Press **Ctrl+F5** to run the project.

The console shows that the runtime found your function. Because you used the `QueueTrigger` attribute in the `ProcessQueueMessage` function, the WebJobs runtime listens for messages in the queue named `queue`. When it finds a new message in this queue, the runtime calls the function, passing in the message string value.

8. Go back to the **Queue** window and refresh it. The message is gone, since it has been processed by your function running locally.

9. Close the console window.

It's now time to publish your WebJobs SDK project to Azure.

## Deploy to Azure

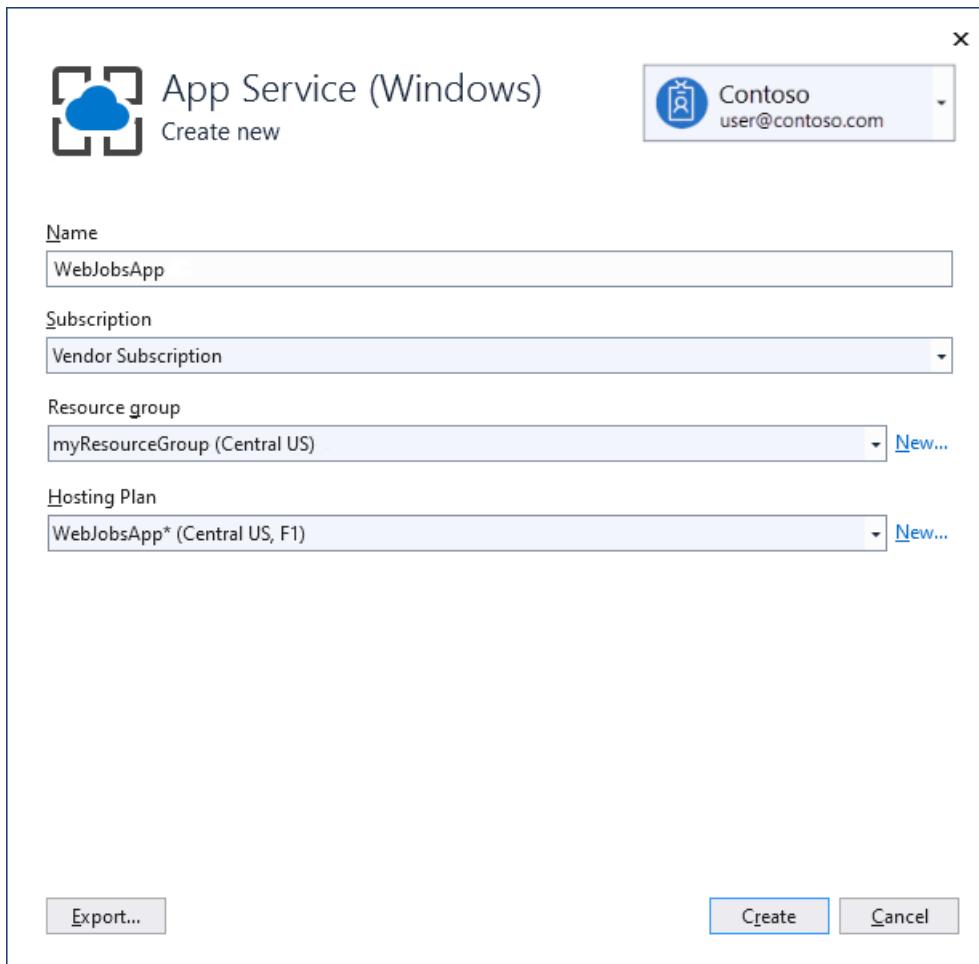
During deployment, you create an app service instance where you'll run your functions. When you publish a .NET Core console app to App Service in Azure, it automatically runs as a WebJob. To learn more about

publishing, see [Develop and deploy WebJobs using Visual Studio](#).

## Create Azure resources

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog box, select **Azure** for **Target**, and then select **Next**.
3. Select **Azure WebJobs** for **Specific target**, and then select **Next**.
4. Above **App Service instances** select the plus (+) button to **Create a new Azure WebJob**.
5. In the **App Service (Windows)** dialog box, use the hosting settings in the following table.

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	Globally unique name	Name that uniquely identifies your new function app.
Subscription	Choose your subscription	The Azure subscription to use.
Resource group	myResourceGroup	Name of the resource group in which to create your function app. Choose <b>New</b> to create a new resource group.
Hosting Plan	App Service plan	An <a href="#">App Service plan</a> specifies the location, size, and features of the web server farm that hosts your app. You can save money when hosting multiple apps by configuring the web apps to share a single App Service plan. App Service plans define the region, instance size, scale count, and SKU (Free, Shared, Basic, Standard, or Premium). Choose <b>New</b> to create a new App Service plan. Free and Basic tiers don't support the Always On option to keep your site running continuously.



6. Select **Create** to create a WebJob and related resources in Azure with these settings and deploy your project code.
7. Select **Finish** to return to the **Publish** page.

### Enable Always On

For a continuous WebJob, you should enable the Always on setting in the site so that your WebJobs run correctly. If you don't enable Always on, the runtime goes idle after a few minutes of inactivity.

1. In the **Publish** page, select the three dots above **Hosting** to show **Hosting profile section actions** and choose **Open in Azure portal**.
2. Under **Settings**, choose **Configuration > General settings**, set **Always on** to **On**, and then select **Save and Continue** to restart the site.

### Publish the project

With the web app created in Azure, it's time to publish the WebJobs project.

1. In the **Publish** page under **Hosting**, select the edit button and change the **WebJob Type** to **Continuous** and select **Save**. This makes sure that the WebJob is running when messages are added to the queue. Triggered WebJobs are typically used only for manual webhooks.
2. Select the **Publish** button at the top right corner of the **Publish** page. When the operation completes, your WebJob is running on Azure.

### Create a storage connection app setting

You need to create the same storage connection string setting in Azure that you used locally in your `appsettings.json` config file. This lets you more securely store the connection string and

1. In your **Publish** profile page, select the three dots above **Hosting** to show **Hosting profile section**

actions and choose **Manage Azure App Service settings**.

2. In **Application settings**, choose **+ Add setting**.
3. In **New app setting name**, type `AzureWebJobsStorage` and select **OK**.
4. In **Remote**, paste in the connection string from your local setting and select **OK**.

The connection string is now set in your app in Azure.

### Trigger the function in Azure

1. Make sure you're not running locally. Close the console window if it's still open. Otherwise, the local instance might be the first to process any queue messages you create.
2. In the **Queue** page in Visual Studio, add a message to the queue as before.
3. Refresh the **Queue** page, and the new message disappears because it has been processed by the function running in Azure.

## Enable Application Insights logging

When the WebJob runs in Azure, you can't monitor function execution by viewing console output. To be able to monitor your WebJob, you should create an associated [Application Insights](#) instance when you publish your project.

### Create an Application Insights instance

1. In your **Publish** profile page, select the three dots above **Hosting** to show **Hosting profile section actions** and choose **Open in Azure Portal**.
2. In the web app under **Settings**, choose **Application Insights**, and select **Turn on Application Insights**.
3. Verify the generated **Resource name** for the instance and the **Location**, and select **Apply**.
4. Under **Settings**, choose **Configuration** and verify that a new `APPINSIGHTS_INSTRUMENTATIONKEY` was created. This key is used to connect your WebJob instance to Application Insights.

To take advantage of [Application Insights](#) logging, you need to update your logging code as well.

### Install the Application Insights extension

1. Get the latest stable version of the [Microsoft.Azure.WebJobs.Logging.ApplicationInsights](#) NuGet package, version 3.x.
2. In the following command, replace `<3_X_VERSION>` with the current version number you found in step 1. Each type of NuGet Package has a unique version number.

```
Install-Package Microsoft.Azure.WebJobs.Logging.ApplicationInsights -Version <3_X_VERSION>
```

3. In the **Package Manager Console**, execute the command with the current version number at the `PM>` entry point.

### Initialize the Application Insights logging provider

Open `Program.cs` and add the following initializer in the `ConfigureLogging` after the call to `AddConsole`:

```
// If the key exists in settings, use it to enable Application Insights.
string instrumentationKey = context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
if (!string.IsNullOrEmpty(instrumentationKey))
{
 b.AddApplicationInsightsWebJobs(o => o.InstrumentationKey = instrumentationKey);
}
```

The `Main` method code should now look like the following example:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.UseEnvironment(EnvironmentName.Development);
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddAzureStorage();
 });
 builder.ConfigureLogging((context, b) =>
 {
 b.AddConsole();

 // If the key exists in settings, use it to enable Application Insights.
 string instrumentationKey = context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
 if (!string.IsNullOrEmpty(instrumentationKey))
 {
 b.AddApplicationInsightsWebJobs(o => o.InstrumentationKey = instrumentationKey);
 }
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

This initializes the Application Insights logging provider with default [filtering](#). When running locally, all information and higher-level logs are written to both the console and Application Insights.

### Republish the project and trigger the function again

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. As before, use **Cloud Explorer** in Visual Studio to create a queue message like you did [earlier](#), except enter *Hello App Insights!* as the message text.
3. In your **Publish** profile page, select the three dots above **Hosting** to show **Hosting profile section actions** and choose **Open in Azure Portal**.
4. In the web app under **Settings**, choose **Application Insights**, and select **View Application Insights data**.
5. Select **Search** and then select **See all data in the last 24 hours**.

The screenshot shows the Azure Application Insights Essentials blade. At the top, there's a search bar with a magnifying glass icon, which is highlighted with a red box. Below the search bar, a message says "NEW - Learn whether users come back to your app with the Retention tool." The main area displays resource details: Resource group (change) is "webapp", Type is "ASP.NET", Location is "West US 2", Subscription name (change) is "Windows Azure MSDN - Visual Studio Ultima...", and Subscription ID is "{subscription ID}".

6. If you don't see the *Hello App Insights!* message, select Refresh periodically for several minutes. Logs don't appear immediately, because it takes a while for the Application Insights client to flush the logs it processes.

The screenshot shows the Azure Application Insights Search blade. The search bar at the top is highlighted with a red box. Below the search bar, there are filters for Time range, Filters, Refresh (which is also highlighted with a red box), Reset, Analytics, and More. The search results show 26 total results between March 26, 2018, 2:07 PM and March 27, 2018, 2:07 PM. A summary chart at the bottom shows the count of different log types: TRACE (21), REQUEST (5), EXCEPTION (0), VIEW (0), EVENT (0), DEPENDENCY (0), and AVAIL (0). The results section lists four log entries, with the first one highlighted with a red box:

- 3/27/2018 2:03:14 PM - TRACE  
Hello Application Insights!  
Device type: PC Severity level: Informational
- 3/27/2018 2:03:14 PM - REQUEST  
ProcessQueueMessage  
Response code: 0 Server response time: 77.33 ms
- 3/27/2018 2:03:14 PM - TRACE  
Job host started  
Device type: PC Severity level: Informational
- 3/27/2018 2:03:14 PM - TRACE

## Add input/output bindings

Bindings simplify code that reads and writes data. Input bindings simplify code that reads data. Output bindings simplify code that writes data.

### Add input binding

Input bindings simplify code that reads data. For this example, the queue message is the name of a blob, which you'll use to find and read a blob in Azure Storage.

1. In *Functions.cs*, replace the `ProcessQueueMessage` method with the following code:

```
public static void ProcessQueueMessage(
 [QueueTrigger("queue")] string message,
 [Blob("container/{queueTrigger}", FileAccess.Read)] Stream myBlob,
 ILogger logger)
{
 logger.LogInformation($"Blob name:{message} \n Size: {myBlob.Length} bytes");
}
```

In this code, `queueTrigger` is a [binding expression](#), which means it resolves to a different value at runtime. At runtime, it has the contents of the queue message.

2. Add a `using`:

```
using System.IO;
```

3. Create a blob container in your storage account.

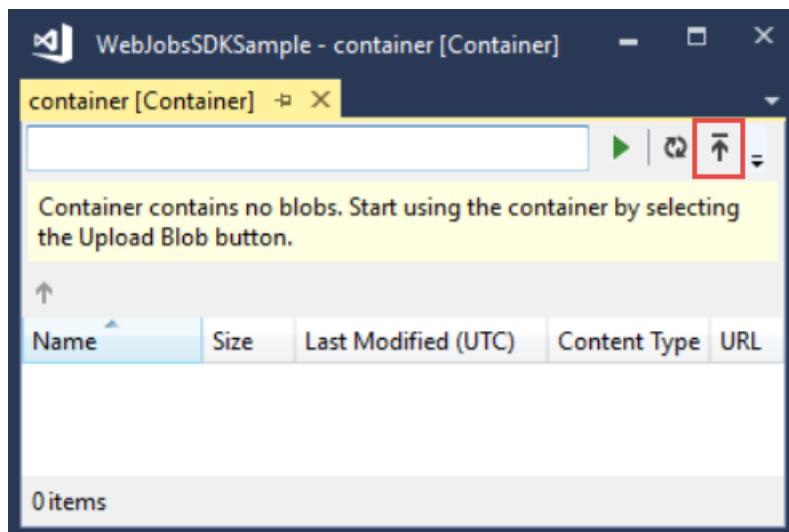
- a. In **Cloud Explorer** in Visual Studio, expand the node for your storage account, right-click **Blobs**, and then select **Create Blob Container**.

- b. In the **Create Blob Container** dialog, enter *container* as the container name, and then select **OK**.

4. Upload the *Program.cs* file to the blob container. (This file is used here as an example; you could upload any text file and create a queue message with the file's name.)

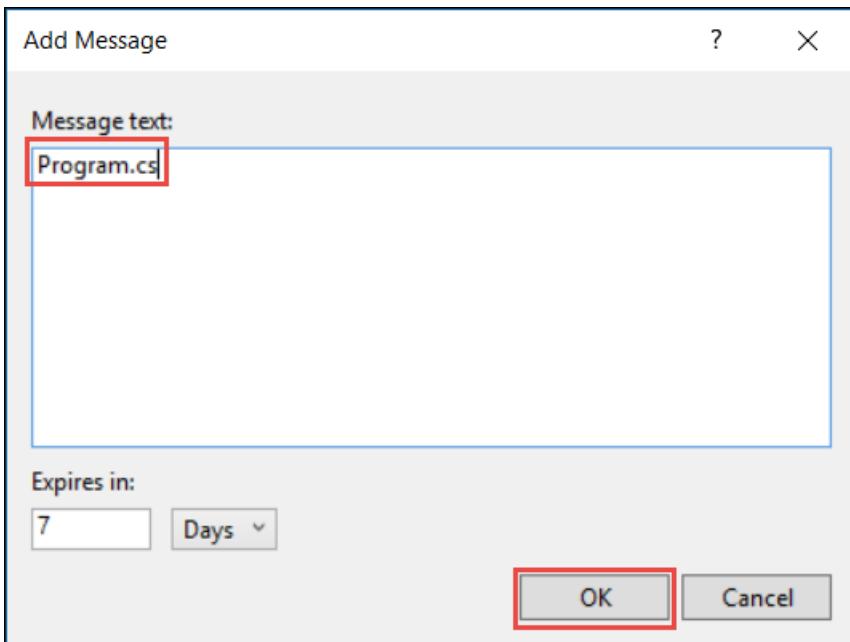
- a. In **Cloud Explorer**, double-click the node for the container you created.

- b. In the **Container** window, select the **Upload** button.



- c. Find and select *Program.cs*, and then select **OK**.

5. Create a queue message in the queue you created earlier, with *Program.cs* as the text of the message.



## 6. Run the project locally.

The queue message triggers the function, which then reads the blob and logs its length. The console output looks like this:

```
Found the following functions:
ConsoleApp1.Functions.ProcessQueueMessage
Job host started
Executing 'Functions.ProcessQueueMessage' (Reason='New queue message detected on 'queue'.',
Id=5a2ac479-de13-4f41-aae9-1361f291ff88)
Blob name:Program.cs
Size: 532 bytes
Executed 'Functions.ProcessQueueMessage' (Succeeded, Id=5a2ac479-de13-4f41-aae9-1361f291ff88)
```

## Add an output binding

Output bindings simplify code that writes data. This example modifies the previous one by writing a copy of the blob instead of logging its size. Blob storage bindings are included in the Azure Storage extension package that we installed previously.

### 1. Replace the `ProcessQueueMessage` method with the following code:

```
public static void ProcessQueueMessage(
 [QueueTrigger("queue")] string message,
 [Blob("container/{queueTrigger}", FileAccess.Read)] Stream myBlob,
 [Blob("container/copy-{queueTrigger}", FileAccess.Write)] Stream outputBlob,
 ILogger logger)
{
 logger.LogInformation($"Blob name:{message} \n Size: {myBlob.Length} bytes");
 myBlob.CopyTo(outputBlob);
}
```

### 2. Create another queue message with `Program.cs` as the text of the message.

### 3. Run the project locally.

The queue message triggers the function, which then reads the blob, logs its length, and creates a new blob. The console output is the same, but when you go to the blob container window and select **Refresh**, you see a new blob named `copy-Program.cs`.

## Republish the project

1. In **Solution Explorer**, right-click the project and select **Publish**.
2. In the **Publish** dialog, make sure that the current profile is selected and then select **Publish**. Results of the publish are detailed in the **Output** window.
3. Verify the function in Azure by again uploading a file to the blob container and adding a message to the queue that is the name of the uploaded file. You see the message get removed from the queue and a copy of the file created in the blob container.

## Next steps

This tutorial showed you how to create, run, and deploy a WebJobs SDK 3.x project.

[Learn more about the WebJobs SDK](#)

# How to use the Azure WebJobs SDK for event-driven background processing

11/2/2021 • 26 minutes to read • [Edit Online](#)

This article provides guidance on how to work with the Azure WebJobs SDK. To get started with WebJobs right away, see [Get started with the Azure WebJobs SDK](#).

## WebJobs SDK versions

These are the key differences between version 3.x and version 2.x of the WebJobs SDK:

- Version 3.x adds support for .NET Core.
- In version 3.x, you'll install the Storage binding extension required by the WebJobs SDK. In version 2.x, the Storage bindings are included in the SDK.
- Visual Studio 2019 tooling for .NET Core (3.x) projects differs from tooling for .NET Framework (2.x) projects. To learn more, see [Develop and deploy WebJobs using Visual Studio - Azure App Service](#).

Several descriptions in this article provide examples for both WebJobs version 3.x and WebJobs version 2.x.

[Azure Functions](#) is built on the WebJobs SDK.

- Azure Functions version 2.x is built on WebJobs SDK version 3.x.
- Azure Functions version 1.x is built on WebJobs SDK version 2.x.

Source code repositories for both Azure Functions and WebJobs SDK use the WebJobs SDK numbering. Several sections of this how-to article link to Azure Functions documentation.

For more information, see [Compare the WebJobs SDK and Azure Functions](#)

## WebJobs host

The host is a runtime container for functions. The Host listens for triggers and calls functions. In version 3.x, the host is an implementation of `IHost`. In version 2.x, you use the `JobHost` object. You create a host instance in your code and write code to customize its behavior.

This is a key difference between using the WebJobs SDK directly and using it indirectly through Azure Functions. In Azure Functions, the service controls the host, and you can't customize the host by writing code. Azure Functions lets you customize host behavior through settings in the `host.json` file. Those settings are strings, not code, and use of these strings limits the kinds of customizations you can do.

### Host connection strings

The WebJobs SDK looks for Azure Storage and Azure Service Bus connection strings in the `local.settings.json` file when you run locally or in the environment of the WebJob when you run in Azure. By default, the WebJobs SDK requires a storage connection string setting with the name `AzureWebJobsStorage`.

Version 2.x of the SDK doesn't require a specific name. Version 2.x lets you use your own names for these connection strings and allows you to store them elsewhere. You can set names in code using the `JobHostConfiguration`, like this:

```

static void Main(string[] args)
{
 var _storageConn = ConfigurationManager
 .ConnectionStrings["MyStorageConnection"].ConnectionString;

 //// Dashboard logging is deprecated; use Application Insights.
 //var _dashboardConn = ConfigurationManager
 // .ConnectionStrings["MyDashboardConnection"].ConnectionString;

 JobHostConfiguration config = new JobHostConfiguration();
 config.StorageConnectionString = _storageConn;
 //config.DashboardConnectionString = _dashboardConn;
 JobHost host = new JobHost(config);
 host.RunAndBlock();
}

```

#### NOTE

Because version 3.x uses the default .NET Core configuration APIs, there is no API to change connection string names. See [Develop and deploy WebJobs using Visual Studio](#)

## Host development settings

You can run the host in development mode to make local development more efficient. Here are some of the settings that automatically change when you run in development mode:

PROPERTY	DEVELOPMENT SETTING
Tracing.ConsoleLevel	TraceLevel.Verbose to maximize log output.
Queues.MaxPollingInterval	A low value to ensure queue methods are triggered immediately.
Singleton.ListenerLockPeriod	15 seconds to aid in rapid iterative development.

The process for enabling development mode depends on the SDK version.

#### Version 3.x

Version 3.x uses the standard ASP.NET Core APIs. Call the `UseEnvironment` method on the `HostBuilder` instance. Pass a string named `development`, as in this example:

```

static async Task Main()
{
 var builder = new HostBuilder();
 builder.UseEnvironment("development");
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}

```

#### Version 2.x

The `JobHostConfiguration` class has a `UseDevelopmentSettings` method that enables development mode. The

following example shows how to use development settings. To make `config.IsDevelopment` return `true` when it runs locally, set a local environment variable named `AzureWebJobsEnv` with the value `Development`.

```
static void Main()
{
 config = new JobHostConfiguration();

 if (config.IsDevelopment)
 {
 config.UseDevelopmentSettings();
 }

 var host = new JobHost(config);
 host.RunAndBlock();
}
```

## Managing concurrent connections (version 2.x)

In version 3.x, the connection limit defaults to infinite connections. If for some reason you need to change this limit, you can use the `MaxConnectionsPerServer` property of the `WinHttpHandler` class.

In version 2.x, you control the number of concurrent connections to a host by using the `ServicePointManager.DefaultConnectionLimit` API. In 2.x, you should increase this value from the default of 2 before starting your WebJobs host.

All outgoing HTTP requests that you make from a function by using `HttpClient` flow through `ServicePointManager`. After you reach the value set in `DefaultConnectionLimit`, `ServicePointManager` starts queueing requests before sending them. Suppose your `DefaultConnectionLimit` is set to 2 and your code makes 1,000 HTTP requests. Initially, only two requests are allowed through to the OS. The other 998 are queued until there's room for them. That means your `HttpClient` might time out because it appears to have made the request, but the request was never sent by the OS to the destination server. So you might see behavior that doesn't seem to make sense: your local `HttpClient` is taking 10 seconds to complete a request, but your service is returning every request in 200 ms.

The default value for ASP.NET applications is `Int32.MaxValue`, and that's likely to work well for WebJobs running in a Basic or higher App Service Plan. WebJobs typically need the **Always On** setting, and that's supported only by Basic and higher App Service Plans.

If your WebJob is running in a Free or Shared App Service Plan, your application is restricted by the App Service sandbox, which currently has a **connection limit of 300**. With an unbound connection limit in `ServicePointManager`, it's more likely that the sandbox connection threshold will be reached and the site will shut down. In that case, setting `DefaultConnectionLimit` to something lower, like 50 or 100, can prevent this from happening and still allow for sufficient throughput.

The setting must be configured before any HTTP requests are made. For this reason, the WebJobs host shouldn't adjust the setting automatically. There could be HTTP requests that occur before the host starts, which could lead to unexpected behavior. The best approach is to set the value immediately in your `Main` method before initializing `JobHost`, as shown here:

```
static void Main(string[] args)
{
 // Set this immediately so that it's used by all requests.
 ServicePointManager.DefaultConnectionLimit = Int32.MaxValue;

 var host = new JobHost();
 host.RunAndBlock();
}
```

# Triggers

WebJobs SDK supports the same set of triggers and binding used by [Azure Functions](#). Please note that in the WebJobs SDK, triggers are function-specific and not related to the WebJob deployment type. WebJobs with event-triggered functions created using the SDK should always be published as a *continuous* WebJob, with *Always on* enabled.

Functions must be public methods and must have one trigger attribute or the `NoAutomaticTrigger` attribute.

## Automatic triggers

Automatic triggers call a function in response to an event. Consider this example of a function that's triggered by a message added to Azure Queue storage. The function responds by reading a blob from Azure Blob storage:

```
public static void Run(
 [QueueTrigger("myqueue-items")] string myQueueItem,
 [Blob("samples-workitems/{queueTrigger}", FileAccess.Read)] Stream myBlob,
 ILogger log)
{
 log.LogInformation($"BlobInput processed blob\n Name:{myQueueItem} \n Size: {myBlob.Length} bytes");
}
```

The `queueTrigger` attribute tells the runtime to call the function whenever a queue message appears in `myqueue-items`. The `Blob` attribute tells the runtime to use the queue message to read a blob in the *samples-workitems* container. The name of the blob item in the `samples-workitems` container is obtained directly from the queue trigger as a binding expression (`{queueTrigger}`).

### NOTE

A web app can time out after 20 minutes of inactivity, and only requests to the actual web app can reset the timer. Viewing the app's configuration in the Azure portal or making requests to the advanced tools site (`https://<app_name>.scm.azurewebsites.net`) doesn't reset the timer. If you set the web app that hosts your job to run continuously, run on a schedule, or use event-driven triggers, enable the **Always on** setting on your web app's Azure Configuration page. The Always on setting helps to make sure that these kinds of WebJobs run reliably. This feature is available only in the Basic, Standard, and Premium [pricing tiers](#).

## Manual triggers

To trigger a function manually, use the `NoAutomaticTrigger` attribute, as shown here:

```
[NoAutomaticTrigger]
public static void CreateQueueMessage(
 ILogger logger,
 string value,
 [Queue("outputqueue")] out string message)
{
 message = value;
 logger.LogInformation("Creating queue message: ", message);
}
```

The process for manually triggering the function depends on the SDK version.

### Version 3.x

```

static async Task Main(string[] args)
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddAzureStorage();
 });
 var host = builder.Build();
 using (host)
 {
 var jobHost = host.Services.GetService(typeof(IJobHost)) as JobHost;
 var inputs = new Dictionary<string, object>
 {
 { "value", "Hello world!" }
 };

 await host.StartAsync();
 await jobHost.CallAsync("CreateQueueMessage", inputs);
 await host.StopAsync();
 }
}

```

#### Version 2.x

```

static void Main(string[] args)
{
 JobHost host = new JobHost();
 host.Call(typeof(Program).GetMethod("CreateQueueMessage"), new { value = "Hello world!" });
}

```

## Input and output bindings

Input bindings provide a declarative way to make data from Azure or third-party services available to your code. Output bindings provide a way to update data. The [Get started](#) article shows an example of each.

You can use a method return value for an output binding by applying the attribute to the method return value. See the example in [Using the Azure Function return value](#).

### Binding types

The process for installing and managing binding types depends on whether you're using version 3.x or version 2.x of the SDK. You can find the package to install for a particular binding type in the "Packages" section of that binding type's Azure Functions [reference article](#). An exception is the Files trigger and binding (for the local file system), which isn't supported by Azure Functions.

#### Version 3.x

In version 3.x, the storage bindings are included in the `Microsoft.Azure.WebJobs.Extensions.Storage` package. Call the `AddAzureStorage` extension method in the `ConfigureWebJobs` method, as shown here:

```

static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddAzureStorage();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}

```

To use other trigger and binding types, install the NuGet package that contains them and call the `Add<binding>` extension method implemented in the extension. For example, if you want to use an Azure Cosmos DB binding, install `Microsoft.Azure.WebJobs.Extensions.CosmosDB` and call `AddCosmosDB`, like this:

```

static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddCosmosDB();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}

```

To use the Timer trigger or the Files binding, which are part of core services, call the `AddTimers` or `AddFiles` extension methods.

#### Version 2.x

These trigger and binding types are included in version 2.x of the `Microsoft.Azure.WebJobs` package:

- Blob storage
- Queue storage
- Table storage

To use other trigger and binding types, install the NuGet package that contains them and call a `Use<binding>` method on the `JobHostConfiguration` object. For example, if you want to use a Timer trigger, install `Microsoft.Azure.WebJobs.Extensions` and call `UseTimers` in the `Main` method, as shown here:

```

static void Main()
{
 config = new JobHostConfiguration();
 config.UseTimers();
 var host = new JobHost(config);
 host.RunAndBlock();
}

```

To use the Files binding, install `Microsoft.Azure.WebJobs.Extensions` and call `UseFiles`.

#### ExecutionContext

WebJobs lets you bind to an `ExecutionContext`. With this binding, you can access the `ExecutionContext` as a parameter in your function signature. For example, the following code uses the context object to access the invocation ID, which you can use to correlate all logs produced by a given function invocation.

```
public class Functions
{
 public static void ProcessQueueMessage([QueueTrigger("queue")] string message,
 ExecutionContext executionContext,
 ILogger logger)
 {
 logger.LogInformation($"{message}\n{executionContext.InvocationId}");
 }
}
```

The process for binding to the `ExecutionContext` depends on your SDK version.

#### Version 3.x

Call the `AddExecutionContextBinding` extension method in the `ConfigureWebJobs` method, as shown here:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddExecutionContextBinding();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

#### Version 2.x

The `Microsoft.Azure.WebJobs.Extensions` package mentioned earlier also provides a special binding type that you can register by calling the `UseCore` method. This binding lets you define an `ExecutionContext` parameter in your function signature, which is enabled like this:

```
class Program
{
 static void Main()
 {
 config = new JobHostConfiguration();
 config.UseCore();
 var host = new JobHost(config);
 host.RunAndBlock();
 }
}
```

## Binding configuration

You can configure the behavior of some triggers and bindings. The process for configuring them depends on the SDK version.

- **Version 3.x:** Set configuration when the `Add<Binding>` method is called in `ConfigureWebJobs`.
- **Version 2.x:** Set configuration by setting properties in a configuration object that you pass in to `JobHost`.

These binding-specific settings are equivalent to settings in the `host.json` project file in Azure Functions.

You can configure the following bindings:

- [Azure CosmosDB trigger](#)
- [Event Hubs trigger](#)
- [Queue storage trigger](#)
- [SendGrid binding](#)
- [Service Bus trigger](#)

#### Azure CosmosDB trigger configuration (version 3.x)

This example shows how to configure the Azure Cosmos DB trigger:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddCosmosDB(a =>
 {
 a.ConnectionMode = ConnectionMode.Gateway;
 a.Protocol = Protocol.Https;
 a.LeaseOptions.LeasePrefix = "prefix1";

 });
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

For more information, see the [Azure CosmosDB binding](#) article.

#### Event Hubs trigger configuration (version 3.x)

This example shows how to configure the Event Hubs trigger:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddEventHubs(a =>
 {
 a.BatchCheckpointFrequency = 5;
 a.EventProcessorOptions.MaxBatchSize = 256;
 a.EventProcessorOptions.PrefetchCount = 512;
 });
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

For more information, see the [Event Hubs binding](#) article.

#### Queue storage trigger configuration

The following examples show how to configure the Queue storage trigger.

## Version 3.x

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddAzureStorage(a => {
 a.BatchSize = 8;
 a.NewBatchThreshold = 4;
 a.MaxDequeueCount = 4;
 a.MaxPollingInterval = TimeSpan.FromSeconds(15);
 });
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

For more information, see the [Queue storage binding](#) article.

## Version 2.x

```
static void Main(string[] args)
{
 JobHostConfiguration config = new JobHostConfiguration();
 config.Queues.BatchSize = 8;
 config.Queues.NewBatchThreshold = 4;
 config.Queues.MaxDequeueCount = 4;
 config.Queues.MaxPollingInterval = TimeSpan.FromSeconds(15);
 JobHost host = new JobHost(config);
 host.RunAndBlock();
}
```

For more information, see the [host.json v1.x reference](#).

## SendGrid binding configuration (version 3.x)

This example shows how to configure the SendGrid output binding:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddSendGrid(a =>
 {
 a.FromAddress.Email = "samples@functions.com";
 a.FromAddress.Name = "Azure Functions";
 });
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

For more information, see the [SendGrid binding](#) article.

## Service Bus trigger configuration (version 3.x)

This example shows how to configure the Service Bus trigger:

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddServiceBus(sbOptions =>
 {
 sbOptions.MessageHandlerOptions.AutoComplete = true;
 sbOptions.MessageHandlerOptions.MaxConcurrentCalls = 16;
 });
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

For more details, see the [Service Bus binding](#) article.

## Configuration for other bindings

Some trigger and binding types define their own custom configuration types. For example, the File trigger lets you specify the root path to monitor, as in the following examples.

### Version 3.x

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 b.AddFiles(a => a.RootPath = @"c:\data\import");
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

### Version 2.x

```
static void Main()
{
 config = new JobHostConfiguration();
 var filesConfig = new FilesConfiguration
 {
 RootPath = @"c:\data\import"
 };
 config.UseFiles(filesConfig);
 var host = new JobHost(config);
 host.RunAndBlock();
}
```

## Binding expressions

In attribute constructor parameters, you can use expressions that resolve to values from various sources. For

example, in the following code, the path for the `BlobTrigger` attribute creates an expression named `filename`. When used for the output binding, `filename` resolves to the name of the triggering blob.

```
public static void CreateThumbnail(
 [BlobTrigger("sample-images/{filename}")] Stream image,
 [Blob("sample-images-sm/{filename}", FileAccess.Write)] Stream imageSmall,
 string filename,
 ILogger logger)
{
 logger.Info($"Blob trigger processing: {filename}");
 // ...
}
```

For more information about binding expressions, see [Binding expressions and patterns](#) in the Azure Functions documentation.

### Custom binding expressions

Sometimes you want to specify a queue name, a blob name or container, or a table name in code rather than hard-coding it. For example, you might want to specify the queue name for the `QueueTrigger` attribute in a configuration file or environment variable.

You can do that by passing a `NameResolver` object in to the `JobHostConfiguration` object. You include placeholders in trigger or binding attribute constructor parameters, and your `NameResolver` code provides the actual values to be used in place of those placeholders. You identify placeholders by surrounding them with percent (%) signs, as shown here:

```
public static void WriteLog([QueueTrigger("%logqueue%")] string logMessage)
{
 Console.WriteLine(logMessage);
}
```

This code lets you use a queue named `logqueue` in the test environment and one named `logqueueprod` in production. Instead of a hard-coded queue name, you specify the name of an entry in the `appSettings` collection.

There's a default `NameResolver` that takes effect if you don't provide a custom one. The default gets values from app settings or environment variables.

Your `NameResolver` class gets the queue name from `appSettings`, as shown here:

```
public class CustomNameResolver : INameResolver
{
 public string Resolve(string name)
 {
 return ConfigurationManager.AppSettings[name].ToString();
 }
}
```

### Version 3.x

You configure the resolver by using dependency injection. These samples require the following `using` statement:

```
using Microsoft.Extensions.DependencyInjection;
```

You add the resolver by calling the `ConfigureServices` extension method on `HostBuilder`, as in this example:

```

static async Task Main(string[] args)
{
 var builder = new HostBuilder();
 var resolver = new CustomNameResolver();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 });
 builder.ConfigureServices(s => s.AddSingleton<INameResolver>(resolver));
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}

```

### Version 2.x

Pass your `NameResolver` class in to the `JobHost` object, as shown here:

```

static void Main(string[] args)
{
 JobHostConfiguration config = new JobHostConfiguration();
 config.NameResolver = new CustomNameResolver();
 JobHost host = new JobHost(config);
 host.RunAndBlock();
}

```

Azure Functions implements `INameResolver` to get values from app settings, as shown in the example. When you use the WebJobs SDK directly, you can write a custom implementation that gets placeholder replacement values from whatever source you prefer.

### Binding at runtime

If you need to do some work in your function before you use a binding attribute like `Queue`, `Blob`, or `Table`, you can use the `IBinder` interface.

The following example takes an input queue message and creates a new message with the same content in an output queue. The output queue name is set by code in the body of the function.

```

public static void CreateQueueMessage(
 [QueueTrigger("inputqueue")] string queueMessage,
 IBinder binder)
{
 string outputQueueName = "outputqueue" + DateTime.Now.Month.ToString();
 QueueAttribute queueAttribute = new QueueAttribute(outputQueueName);
 CloudQueue outputQueue = binder.Bind<CloudQueue>(queueAttribute);
 outputQueue.AddMessageAsync(new CloudQueueMessage(queueMessage));
}

```

For more information, see [Binding at runtime](#) in the Azure Functions documentation.

### Binding reference information

The Azure Functions documentation provides reference information about each binding type. You'll find the following information in each binding reference article. (This example is based on Storage queue.)

- **Packages.** The package you need to install to include support for the binding in a WebJobs SDK project.
- **Examples.** Code samples. The C# class library example applies to the WebJobs SDK. Just omit the `FunctionName` attribute.
- **Attributes.** The attributes to use for the binding type.

- **Configuration.** Explanations of the attribute properties and constructor parameters.
- **Usage.** The types you can bind to and information about how the binding works. For example: polling algorithm, poison queue processing.

#### NOTE

The HTTP, Webhooks, and Event Grid bindings are supported only by Azure Functions, not by the WebJobs SDK.

For a full list of bindings supported in Azure Functions runtime, see [Supported bindings](#).

## Attributes for Disable, Timeout, and Singleton

With these attributes, you can control function triggering, cancel functions, and ensure that only one instance of a function runs.

### Disable attribute

The `Disable` attribute lets you control whether a function can be triggered.

In the following example, if the app setting `Disable_TestJob` has a value of `1` or `True` (case insensitive), the function won't run. In that case, the runtime creates a log message *Function 'Functions.TestJob' is disabled*.

```
[Disable("Disable_TestJob")]
public static void TestJob([QueueTrigger("testqueue2")] string message)
{
 Console.WriteLine("Function with Disable attribute executed!");
}
```

When you change app setting values in the Azure portal, the WebJob restarts to pick up the new setting.

The attribute can be declared at the parameter, method, or class level. The setting name can also contain binding expressions.

### Timeout attribute

The `Timeout` attribute causes a function to be canceled if it doesn't finish within a specified amount of time. In the following example, the function would run for one day without the `Timeout` attribute. `Timeout` causes the function to be canceled after 15 seconds.

```
[Timeout("00:00:15")]
public static async Task TimeoutJob(
 [QueueTrigger("testqueue2")] string message,
 CancellationToken token,
 TextWriter log)
{
 await log.WriteLineAsync("Job starting");
 await Task.Delay(TimeSpan.FromDays(1), token);
 await log.WriteLineAsync("Job completed");
}
```

You can apply the `Timeout` attribute at the class or method level, and you can specify a global timeout by using `JobHostConfiguration.FunctionTimeout`. Class-level or method-level timeouts override global timeouts.

### Singleton attribute

The `Singleton` attribute ensures that only one instance of a function runs, even when there are multiple instances of the host web app. The `Singleton` attribute uses [distributed locking](#) to ensure that one instance runs.

In this example, only a single instance of the `ProcessImage` function runs at any given time:

```
[Singleton]
public static async Task ProcessImage([BlobTrigger("images")] Stream image)
{
 // Process the image.
}
```

#### SingletonMode.Listener

Some triggers have built-in support for concurrency management:

- **QueueTrigger**. Set `JobHostConfiguration.Queues.BatchSize` to `1`.
- **ServiceBusTrigger**. Set `ServiceBusConfiguration.MessageOptions.MaxConcurrentCalls` to `1`.
- **FileTrigger**. Set `FileProcessor.MaxDegreeOfParallelism` to `1`.

You can use these settings to ensure that your function runs as a singleton on a single instance. To ensure that only a single instance of the function is running when the web app scales out to multiple instances, apply a listener-level singleton lock on the function (`[Singleton(mode = SingletonMode.Listener)]`). Listener locks are acquired when the JobHost starts. If three scaled-out instances all start at the same time, only one of the instances acquires the lock and only one listener starts.

#### NOTE

See this [GitHub Repo](#) to learn more about how the `SingletonMode.Function` works.

#### Scope values

You can specify a *scope expression/value* on a singleton. The expression/value ensures that all executions of the function at a specific scope will be serialized. Implementing more granular locking in this way can allow for some level of parallelism for your function while serializing other invocations as dictated by your requirements. For example, in the following code, the scope expression binds to the `Region` value of the incoming message. When the queue contains three messages in regions East, East, and West, the messages that have region East are run serially. The message with region West is run in parallel with those in region East.

```
[Singleton("{Region}")]
public static async Task ProcessWorkItem([QueueTrigger("workitems")] WorkItem workItem)
{
 // Process the work item.

 public class WorkItem
 {
 public int ID { get; set; }
 public string Region { get; set; }
 public int Category { get; set; }
 public string Description { get; set; }
 }
}
```

#### SingletonScope.Host

The default scope for a lock is `SingletonScope.Function`, meaning the lock scope (the blob lease path) is tied to the fully qualified function name. To lock across functions, specify `SingletonScope.Host` and use a scope ID name that's the same across all functions that you don't want to run simultaneously. In the following example, only one instance of `AddItem` or `RemoveItem` runs at a time:

```

[Singleton("ItemsLock", SingletonScope.Host)]
public static void AddItem([QueueTrigger("add-item")] string message)
{
 // Perform the add operation.
}

[Singleton("ItemsLock", SingletonScope.Host)]
public static void RemoveItem([QueueTrigger("remove-item")] string message)
{
 // Perform the remove operation.
}

```

## Viewing lease blobs

The WebJobs SDK uses [Azure blob leases](#) under the covers to implement distributed locking. The lease blobs used by Singleton can be found in the `azure-webjobs-host` container in the `AzureWebJobsStorage` storage account under the path "locks". For example, the lease blob path for the first `ProcessImage` example shown earlier might be `locks/061851c758f04938a4426aa9ab3869c0/WebJobs.Functions.ProcessImage`. All paths include the JobHost ID, in this case `061851c758f04938a4426aa9ab3869c0`.

## Async functions

For information about how to code async functions, see the [Azure Functions documentation](#).

## Cancellation tokens

For information about how to handle cancellation tokens, see the Azure Functions documentation on [cancellation tokens and graceful shutdown](#).

## Multiple instances

If your web app runs on multiple instances, a continuous WebJob runs on each instance, listening for triggers and calling functions. The various trigger bindings are designed to efficiently share work collaboratively across instances, so that scaling out to more instances allows you to handle more load.

While some triggers may result in double-processing, queue and blob storage triggers automatically prevent a function from processing a queue message or blob more than once. For more information, see [Designing for identical input](#) in the Azure Functions documentation.

The timer trigger automatically ensures that only one instance of the timer runs, so you don't get more than one function instance running at a given scheduled time.

If you want to ensure that only one instance of a function runs even when there are multiple instances of the host web app, you can use the `Singleton` attribute.

## Filters

Function Filters (preview) provide a way to customize the WebJobs execution pipeline with your own logic. Filters are similar to [ASP.NET Core filters](#). You can implement them as declarative attributes that are applied to your functions or classes. For more information, see [Function Filters](#).

## Logging and monitoring

We recommend the logging framework that was developed for ASP.NET. The [Get started](#) article shows how to use it.

## Log filtering

Every log created by an `ILogger` instance has an associated `Category` and `Level`. `LogLevel` is an enumeration, and the integer code indicates relative importance:

LOGLEVEL	CODE
Trace	0
Debug	1
Information	2
Warning	3
Error	4
Critical	5
None	6

You can independently filter each category to a particular `LogLevel`. For example, you might want to see all logs for blob trigger processing but only `Error` and higher for everything else.

### Version 3.x

Version 3.x of the SDK relies on the filtering built into .NET Core. The `LogCategories` class lets you define categories for specific functions, triggers, or users. It also defines filters for specific host states, like `Startup` and `Results`. This enables you to fine-tune the logging output. If no match is found within the defined categories, the filter falls back to the `Default` value when deciding whether to filter the message.

`LogCategories` requires the following using statement:

```
using Microsoft.Azure.WebJobs.Logging;
```

The following example constructs a filter that, by default, filters all logs at the `Warning` level. The `Function` and `results` categories (equivalent to `Host.Results` in version 2.x) are filtered at the `Error` level. The filter compares the current category to all registered levels in the `LogCategories` instance and chooses the longest match. This means that the `Debug` level registered for `Host.Triggers` matches `Host.Triggers.Queue` or `Host.Triggers.Blob`. This allows you to control broader categories without needing to add each one.

```

static async Task Main(string[] args)
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 });
 builder.ConfigureLogging(logging =>
 {
 logging.SetMinimumLevel(LogLevel.Warning);
 logging.AddFilter("Function", LogLevel.Error);
 logging.AddFilter(LogCategories.CreateFunctionCategory("MySpecificFunctionName"),
 LogLevel.Debug);
 logging.AddFilter(LogCategories.Results, LogLevel.Error);
 logging.AddFilter("Host.Triggers", LogLevel.Debug);
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}

```

### Version 2.x

In version 2.x of the SDK, you use the `LogCategoryFilter` class to control filtering. The `LogCategoryFilter` has a `Default` property with an initial value of `Information`, meaning that any messages at the `Information`, `Warning`, `Error`, or `Critical` levels are logged, but any messages at the `Debug` or `Trace` levels are filtered away.

As with `LogCategories` in version 3.x, the `CategoryLevels` property allows you to specify log levels for specific categories so you can fine-tune the logging output. If no match is found within the `CategoryLevels` dictionary, the filter falls back to the `Default` value when deciding whether to filter the message.

The following example constructs a filter that by default filters all logs at the `Warning` level. The `Function` and `Host.Results` categories are filtered at the `Error` level. The `LogCategoryFilter` compares the current category to all registered `CategoryLevels` and chooses the longest match. So the `Debug` level registered for `Host.Triggers` will match `Host.Triggers.Queue` or `Host.Triggers.Blob`. This allows you to control broader categories without needing to add each one.

```

var filter = new LogCategoryFilter();
filter.DefaultLevel = LogLevel.Warning;
filter.CategoryLevels[LogCategories.Function] = LogLevel.Error;
filter.CategoryLevels[LogCategories.Results] = LogLevel.Error;
filter.CategoryLevels["Host.Triggers"] = LogLevel.Debug;

config.LoggerFactory = new LoggerFactory()
 .AddApplicationInsights(instrumentationKey, filter.Filter)
 .AddConsole(filter.Filter);

```

### Custom telemetry for Application Insights

The process for implementing custom telemetry for [Application Insights](#) depends on the SDK version. To learn how to configure Application Insights, see [Add Application Insights logging](#).

### Version 3.x

Because version 3.x of the WebJobs SDK relies on the .NET Core generic host, a custom telemetry factory is no longer provided. But you can add custom telemetry to the pipeline by using dependency injection. The examples in this section require the following `using` statements:

```
using Microsoft.ApplicationInsights.Extensibility;
using Microsoft.ApplicationInsights.Channel;
```

The following custom implementation of `ITelemetryInitializer` lets you add your own `ITelemetry` to the default `TelemetryConfiguration`.

```
internal class CustomTelemetryInitializer : ITelemetryInitializer
{
 public void Initialize(ITelemetry telemetry)
 {
 // Do something with telemetry.
 }
}
```

Call `ConfigureServices` in the builder to add your custom `ITelemetryInitializer` to the pipeline.

```
static async Task Main()
{
 var builder = new HostBuilder();
 builder.ConfigureWebJobs(b =>
 {
 b.AddAzureStorageCoreServices();
 });
 builder.ConfigureLogging((context, b) =>
 {
 // Add logging providers.
 b.AddConsole();

 // If this key exists in any config, use it to enable Application Insights.
 string appInsightsKey = context.Configuration["APPINSIGHTS_INSTRUMENTATIONKEY"];
 if (!string.IsNullOrEmpty(appInsightsKey))
 {
 // This uses the options callback to explicitly set the instrumentation key.
 b.AddApplicationInsights(o => o.InstrumentationKey = appInsightsKey);
 }
 });
 builder.ConfigureServices(services =>
 {
 services.AddSingleton<ITelemetryInitializer, CustomTelemetryInitializer>();
 });
 var host = builder.Build();
 using (host)
 {
 await host.RunAsync();
 }
}
```

When the `TelemetryConfiguration` is constructed, all registered types of `ITelemetryInitializer` are included. To learn more, see [Application Insights API for custom events and metrics](#).

In version 3.x, you no longer have to flush the `TelemetryClient` when the host stops. The .NET Core dependency injection system automatically disposes of the registered `ApplicationInsightsLoggerProvider`, which flushes the `TelemetryClient`.

#### Version 2.x

In version 2.x, the `TelemetryClient` created internally by the Application Insights provider for the WebJobs SDK uses `ServerTelemetryChannel`. When the Application Insights endpoint is unavailable or throttling incoming requests, this channel [saves requests in the web app's file system and resubmits them later](#).

The `TelemetryClient` is created by a class that implements `ITelemetryClientFactory`. By default, this is the

```
DefaultTelemetryClientFactory
```

.

If you want to modify any part of the Application Insights pipeline, you can supply your own `ITelemetryClientFactory`, and the host will use your class to construct a `TelemetryClient`. For example, this code overrides `DefaultTelemetryClientFactory` to modify a property of `ServerTelemetryChannel`:

```
private class CustomTelemetryClientFactory : DefaultTelemetryClientFactory
{
 public CustomTelemetryClientFactory(string instrumentationKey, Func<string, LogLevel, bool> filter)
 : base(instrumentationKey, new SamplingPercentageEstimatorSettings(), filter)
 {
 }

 protected override ITelemetryChannel CreateTelemetryChannel()
 {
 ServerTelemetryChannel channel = new ServerTelemetryChannel();

 // Change the default from 30 seconds to 15 seconds.
 channel.MaxTelemetryBufferDelay = TimeSpan.FromSeconds(15);

 return channel;
 }
}
```

The `SamplingPercentageEstimatorSettings` object configures [adaptive sampling](#). This means that in certain high-volume scenarios, Applications Insights sends a selected subset of telemetry data to the server.

After you create the telemetry factory, you pass it in to the Application Insights logging provider:

```
var clientFactory = new CustomTelemetryClientFactory(instrumentationKey, filter.Filter);

config.LoggerFactory = new LoggerFactory()
 .AddApplicationInsights(clientFactory);
```

## Next steps

This article has provided code snippets that show how to handle common scenarios for working with the WebJobs SDK. For complete samples, see [azure-webjobs-sdk-samples](#).

# Availability Zone support for public multi-tenant App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

Microsoft Azure App Service can be deployed into [Availability Zones \(AZ\)](#) which enables [high availability](#) for your apps. This architecture is also known as zone redundancy.

An app lives in an App Service plan (ASP), and the App Service plan exists in a single scale unit. When an App Service is configured to be zone redundant, the platform automatically spreads the VM instances in the App Service plan across all three zones in the selected region. If a capacity larger than three is specified and the number of instances is divisible by three, the instances will be spread evenly. Otherwise, instance counts beyond  $3 \times N$  will get spread across the remaining one or two zones.

## Requirements

Zone redundancy is a property of the App Service plan. The following are the current requirements/limitations for enabling zone redundancy:

- Both Windows and Linux are supported
- Requires either **Premium v2** or **Premium v3** App Service plans
- Minimum instance count of three
  - The platform will enforce this minimum count behind the scenes if you specify an instance count fewer than three
- Can be enabled in any of the following regions:
  - West US 2
  - West US 3
  - Central US
  - East US
  - East US 2
  - Canada Central
  - Brazil South
  - North Europe
  - West Europe
  - Germany West Central
  - France Central
  - UK South
  - Japan East
  - Southeast Asia
  - Australia East
- Zone redundancy can only be specified when creating a **new** App Service plan
  - Currently you can't convert a pre-existing App Service plan. See next bullet for details on how to create a new App Service plan that supports zone redundancy.
- Zone redundancy is only supported in the newer portion of the App Service footprint
  - Currently if you're running on Pv3, then it is possible that you're already on a footprint that supports zone redundancy. In this scenario, you can create a new App Service plan and specify zone redundancy when creating the new App Service plan.

- If you aren't using Pv3 or a scale unit that supports zone redundancy, are in an unsupported region, or are unsure, follow the steps below:
  - Create a new resource group in a region that is supported
    - This ensures the App Service control plane can find a scale unit in the selected region that supports zone redundancy
  - Create a new App Service plan (and app) in a region of your choice using the **new** resource group
  - Ensure the `zoneRedundant` property (described below) is set to true when creating the new App Service plan
- Must be created using [Azure Resource Manager \(ARM\) templates](#)

In the case when a zone goes down, the App Service platform will detect lost instances and automatically attempt to find new replacement instances. If you also have autoscale configured, and if it decides more instances are needed, autoscale will also issue a request to App Service to add more instances (autoscale behavior is independent of App Service platform behavior). It's important to note there's no guarantee that requests for additional instances in a zone-down scenario will succeed since back filling lost instances occurs on a best-effort basis. The recommended solution is to provision your App Service plans to account for losing a zone as described in the next section of this article.

Applications deployed in an App Service plan enabled for zone redundancy will continue to run and serve traffic even if other zones in the same region suffer an outage. However it's possible that non-runtime behaviors including App Service plan scaling, application creation, application configuration, and application publishing may still be impacted from an outage in other Availability Zones. Zone redundancy for App Service plans only ensures continued uptime for deployed applications.

When the App Service platform allocates instances to a zone redundant App Service plan, it uses [best effort zone balancing offered by the underlying Azure Virtual Machine Scale Sets](#). An App Service plan will be "balanced" if each zone has either the same number of VMs, or +/- 1 VM in all of the other zones used by the App Service plan.

## How to Deploy a Zone Redundant App Service

Currently, you need to use an ARM template to create a zone redundant App Service. Once created via an ARM template, the App Service plan can be viewed and interacted with via the Azure portal and CLI tooling. An ARM template is only needed for the initial creation of the App Service plan.

The only changes needed in an ARM template to specify a zone redundant App Service are the new `zoneRedundant` property (required) and optionally the App Service plan instance count (`capacity`) on the [Microsoft.Web/serverfarms](#) resource. If you don't specify a capacity, the platform defaults to three. The `zoneRedundant` property should be set to `true` and `capacity` should be set based on the workload requirement, but no less than three. A good rule of thumb to choose capacity is to ensure sufficient instances for the application such that losing one zone of instances leaves sufficient capacity to handle expected load.

### TIP

To decide instance capacity, you can use the following calculation:

Since the platform spreads VMs across 3 zones and you need to account for at least the failure of 1 zone, multiply peak workload instance count by a factor of  $\text{zones}/(\text{zones}-1)$ , or  $3/2$ . For example, if your typical peak workload requires 4 instances, you should provision 6 instances:  $(2/3 * 6 \text{ instances}) = 4 \text{ instances}$ .

The ARM template snippet below shows the new `zoneRedundant` property and `capacity` specification.

```
"resources": [
 {
 "type": "Microsoft.Web/serverfarms",
 "apiVersion": "2018-02-01",
 "name": "your-appserviceplan-name-here",
 "location": "West US 3",
 "sku": {
 "name": "P1v3",
 "tier": "PremiumV3",
 "size": "P1v3",
 "family": "Pv3",
 "capacity": 3
 },
 "kind": "app",
 "properties": {
 "zoneRedundant": true
 }
 }
]
```

## Next steps

[Learn how to create and deploy ARM templates](#)

[ARM Quickstart Templates](#)

# Create App Service app using Bicep

11/2/2021 • 3 minutes to read • [Edit Online](#)

Get started with [Azure App Service](#) by deploying an app to the cloud using a [Bicep](#) file and [Azure CLI](#) in Cloud Shell. Because you use a free App Service tier, you incur no costs to complete this quickstart.

Bicep is a domain-specific language (DSL) that uses declarative syntax to deploy Azure resources. It provides concise syntax, reliable type safety, and support for code reuse. You can use Bicep instead of JSON to develop your Azure Resource Manager templates ([ARM templates](#)). The JSON syntax to create an ARM template can be verbose and require complicated expressions. Bicep syntax reduces that complexity and improves the development experience. Bicep is a transparent abstraction over ARM template JSON and doesn't lose any of the JSON template capabilities. During deployment, Bicep CLI transpiles a Bicep file into ARM template JSON.

## Prerequisites

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

To effectively create resources with Bicep, you'll need to set up a Bicep [development environment](#). The Bicep extension for [Visual Studio Code](#) provides language support and resource autocompletion. The extension helps you create and validate Bicep files and is recommended for those developers that will create resources using Bicep after completing this quickstart.

## Review the template

The template used in this quickstart is shown below. It deploys an App Service plan and an App Service app on Linux and a sample Node.js "Hello World" app from the [Azure Samples](#) repo.

```

param webAppName string = uniqueString(resourceGroup().id) // Generate unique String for web app name
param sku string = 'F1' // The SKU of App Service Plan
param linuxFxVersion string = 'node|14-lts' // The runtime stack of web app
param location string = resourceGroup().location // Location for all resources
param repositoryUrl string = 'https://github.com/Azure-Samples/nodejs-docs-hello-world'
param branch string = 'master'
var appServicePlanName = toLower('AppServicePlan-${webAppName}')
var webSiteName = toLower('wapp-${webAppName}')
resource appServicePlan 'Microsoft.Web/serverfarms@2020-06-01' = {
 name: appServicePlanName
 location: location
 properties: {
 reserved: true
 }
 sku: {
 name: sku
 }
 kind: 'linux'
}
resource appService 'Microsoft.Web/sites@2020-06-01' = {
 name: webSiteName
 location: location
 properties: {
 serverFarmId: appServicePlan.id
 siteConfig: {
 linuxFxVersion: linuxFxVersion
 }
 }
}
resource srcControls 'Microsoft.Web/sites/sourcecontrols@2021-01-01' = {
 name: '${appService.name}/web'
 properties: {
 repoUrl: repositoryUrl
 branch: branch
 isManualIntegration: true
 }
}

```

Three Azure resources are defined in the template:

- **Microsoft.Web/serverfarms**: create an App Service plan.
- **Microsoft.Web/sites**: create an App Service app.
- **Microsoft.Web/sites/sourcecontrols**: create an external git deployment configuration.

This template contains several parameters that are predefined for your convenience. See the table below for parameter defaults and their descriptions:

PARAMETERS	TYPE	DEFAULT VALUE	DESCRIPTION
webAppName	string	"webApp- <uniqueString> "	App name
location	string	"[resourceGroup().location]"	App region
sku	string	"F1"	Instance size
linuxFxVersion	string	"NODE 14-LTS"	"Programming language stack   Version"

PARAMETERS	TYPE	DEFAULT VALUE	DESCRIPTION
repositoryUrl	string	"https://github.com/Azure-Samples/nodejs-docs-hello-world"	External Git repo (optional)
branch	string	"master"	Default branch for code sample

## Deploy the template

Copy and paste the template to your preferred editor/IDE and save the file to your local working directory.

Azure CLI is used here to deploy the template. You can also use the Azure portal, Azure PowerShell, and REST API. To learn other deployment methods, see [Bicep Deployment Commands](#).

The following code creates a resource group, an App Service plan, and a web app. A default resource group, App Service plan, and location have been set for you. Replace `<app-name>` with a globally unique app name (valid characters are `a-z`, `0-9`, and `-`).

Open up a terminal where the Azure CLI is installed and run the code below to create a Node.js app on Linux.

```
az group create --name myResourceGroup --location "southcentralus" &&
az deployment group create --resource-group myResourceGroup --template-file <path-to-template>
```

To deploy a different language stack, update `linuxFxVersion` with appropriate values. Samples are shown below.

To show current versions, run the following command in the Cloud Shell:

```
az webapp config show --resource-group myResourceGroup --name <app-name> --query linuxFxVersion
```

LANGUAGE	EXAMPLE
.NET	linuxFxVersion="DOTNETCORE 3.0"
PHP	linuxFxVersion="PHP 7.4"
Node.js	linuxFxVersion="NODE 10.15"
Java	linuxFxVersion="JAVA 1.8  TOMCAT 9.0"
Python	linuxFxVersion="PYTHON 3.7"
Ruby	linuxFxVersion="RUBY 2.6"

## Validate the deployment

Browse to `http://<app_name>.azurewebsites.net/` and verify it's been created.

## Clean up resources

When no longer needed, [delete the resource group](#).

## Next steps

[Bicep documentation](#)

[Bicep samples for Azure App Service](#)

# Create App Service app using a Terraform template

11/2/2021 • 3 minutes to read • [Edit Online](#)

Get started with [Azure App Service](#) by deploying an app to the cloud using [Terraform](#). Because you use a free App Service tier, you incur no costs to complete this quickstart.

Terraform allows you to define and create complete infrastructure deployments in Azure. You build Terraform templates in a human-readable format that create and configure Azure resources in a consistent, reproducible manner. This article shows you how to create a Windows app with Terraform.

## Prerequisites

Azure subscription: If you don't have an Azure subscription, create a [free account](#) before you begin.

Configure Terraform: If you haven't already done so, configure Terraform using one of the following options:

- [Configure Terraform in Azure Cloud Shell with Bash](#)
- [Configure Terraform in Azure Cloud Shell with PowerShell](#)
- [Configure Terraform in Windows with Bash](#)
- [Configure Terraform in Windows with PowerShell](#)

The Azure Terraform Visual Studio Code extension enables you to work with Terraform from the editor. With this extension, you can author, test, and run Terraform configurations. The extension also supports resource graph visualization. See [this guide](#) for configuring the Azure Terraform Visual Studio Code extension.

## Review the template

The template used in this quickstart is shown below. It deploys an App Service plan and an App Service app on Windows and a sample Node.js "Hello World" app from the [Azure Samples](#) repo.

```

Configure the Azure provider
terraform {
 required_providers {
 azurerm = {
 source = "hashicorp/azurerm"
 version = "~> 2.65"
 }
 }
 required_version = ">= 0.14.9"
}

provider "azurerm" {
 features {}
}

Generate a random integer to create a globally unique name
resource "random_integer" "ri" {
 min = 10000
 max = 99999
}

Create the resource group
resource "azurerm_resource_group" "rg" {
 name = "myResourceGroup-${random_integer.ri.result}"
 location = "eastus"
}

Create the Linux App Service Plan
resource "azurerm_app_service_plan" "appserviceplan" {
 name = "webapp-asp-${random_integer.ri.result}"
 location = azurerm_resource_group.rg.location
 resource_group_name = azurerm_resource_group.rg.name
 sku {
 tier = "Free"
 size = "F1"
 }
}

Create the web app, pass in the App Service Plan ID, and deploy code from a public GitHub repo
resource "azurerm_app_service" "webapp" {
 name = "webapp-${random_integer.ri.result}"
 location = azurerm_resource_group.rg.location
 resource_group_name = azurerm_resource_group.rg.name
 app_service_plan_id = azurerm_app_service_plan.appserviceplan.id
 source_control {
 repo_url = "https://github.com/Azure-Samples/nodejs-docs-hello-world"
 branch = "master"
 manual_integration = true
 use_mercurial = false
 }
}

```

Three Azure resources and one subresource are defined in the template. Links to the [Azure Provider Terraform Registry](#) are given below for further details and usage information:

- [Microsoft.Resources/resourcegroups](#): create a Resource Group if one doesn't already exist.
  - [azurerm\\_resource\\_group](#)
- [Microsoft.Web/serverfarms](#): create an App Service plan.
  - [azurerm\\_app\\_service\\_plan](#)
- [Microsoft.Web/sites](#): create an App Service app.
  - [azurerm\\_app\\_service](#)
- [Microsoft.Web/sites/sourcecontrols](#): create an external git deployment configuration.
  - [source\\_control](#)

For further information on how to construct Terraform templates, have a look at the [Terraform Learn documentation](#).

## Implement the Terraform code

Terraform provides many features for managing, building, deploying, and updating infrastructure. The steps below will just guide you through deploying and destroying your resources. The [Terraform Learn documentation](#) and [Terraform on Azure documentation](#) go into more detail and should be reviewed if Terraform is part of your Azure infrastructure strategy.

1. Create a directory in which to test and run the sample Terraform code and make it the current directory.

```
mkdir appservice_tf_quickstart
cd appservice_tf_quickstart
```

2. Create a file named `main.tf` and insert the above code.

```
code main.tf
```

3. Initialize Terraform.

```
terraform init
```

4. Create the Terraform plan.

```
terraform plan
```

5. Provision the resources that are defined in the `main.tf` configuration file (Confirm the action by entering `yes` at the prompt).

```
terraform apply
```

## Validate the deployment

1. On the main menu of the Azure portal, select **Resource groups** and navigate to the resource group you created with the above template. It will be named "myResourceGroup-" followed by a string of random integers.
2. You now see all the resources that Terraform has created (an App Service and an App Service Plan).
3. Select the **App Service** and navigate to the url to verify your site has been created properly. Instead, you can just browse to `http://<app_name>.azurewebsites.net/` where app name is "webapp-" followed by that same string of random integers from the resource group.

## Clean up resources

When no longer needed, either [delete the resource group](#) or head back to your terminal/command line and execute `terraform destroy` to delete all resources associated with this quickstart.

### NOTE

You can find more Azure App Service Terraform samples [here](#). You can find even more Terraform samples across all of the Azure services [here](#).

## Next steps

[Learn more about using Terraform in Azure](#)

[Terraform samples for Azure App Service](#)

# Monitoring App Service data reference

11/2/2021 • 2 minutes to read • [Edit Online](#)

This reference applies to the use of Azure Monitor for monitoring App Service. See [Monitoring App Service](#) for details on collecting and analyzing monitoring data for App Service.

## Metrics

This section lists all the automatically collected platform metrics collected for App Service.

METRIC TYPE	RESOURCE PROVIDER / TYPE NAMESPACE AND LINK TO INDIVIDUAL METRICS
App Service Plans	<a href="#">Microsoft.Web/serverfarms</a>
Web apps	<a href="#">Microsoft.Web/sites</a>
Staging slots	<a href="#">Microsoft.Web/sites/slots</a>
App Service Environment	<a href="#">Microsoft.Web/hostingEnvironments</a>
App Service Environment Front-end	<a href="#">Microsoft.Web/hostingEnvironments/multiRolePools</a>

For more information, see a list of [all platform metrics supported in Azure Monitor](#).

## Metric Dimensions

For more information on what metric dimensions are, see [Multi-dimensional metrics](#).

App Service doesn't have any metrics that contain dimensions.

## Resource logs

This section lists the types of resource logs you can collect for App Service.

LOG TYPE	WINDOWS	WINDOWS CONTAINER	LINUX	LINUX CONTAINER	DESCRIPTION
AppServiceConsoleLogs	Java SE & Tomcat	Yes	Yes	Yes	Standard output and standard error
AppServiceHTTPLogs	Yes	Yes	Yes	Yes	Web server logs
AppServiceEnvironmentPlatformLogs	Yes	N/A	Yes	Yes	App Service Environment: scaling, configuration changes, and status logs

LOG TYPE	WINDOWS	WINDOWS CONTAINER	LINUX	LINUX CONTAINER	DESCRIPTION
AppServiceAuditLogs	Yes	Yes	Yes	Yes	Login activity via FTP and Kudu
AppServiceFileAuditLogs	Yes	Yes	TBA	TBA	File changes made to the site content; <b>only available for Premium tier and above</b>
AppServiceAppLogs	ASP.NET	ASP.NET	Java SE & Tomcat Images <sup>1</sup>	Java SE & Tomcat Blessed Images <sup>1</sup>	Application logs
AppServiceIPSecAuditLogs	Yes	Yes	Yes	Yes	Requests from IP Rules
AppServicePlatformLogs	TBA	Yes	Yes	Yes	Container operation logs
AppServiceAntivirusScanAuditLogs	Yes	Yes	Yes	Yes	<a href="#">Anti-virus scan logs</a> using Microsoft Defender; <b>only available for Premium tier</b>

<sup>1</sup> For Java SE apps, add "\$WEBSITE\_AZMON\_PREVIEW\_ENABLED" to the app settings and set it to 1 or to true.

For reference, see a list of [all resource logs category types supported in Azure Monitor](#).

## Azure Monitor Logs tables

Azure App Service uses Kusto tables from Azure Monitor Logs. You can query these tables with Log analytics. For a list of App Service tables used by Kusto, see the [Azure Monitor Logs table reference - App Service tables](#).

## Activity log

The following table lists common operations related to App Service that may be created in the Activity log. This is not an exhaustive list.

OPERATION	DESCRIPTION
Create or Update Web App	App was created or updated
Delete Web App	App was deleted
Create Web App Backup	Backup of app
Get Web App Publishing Profile	Download of publishing profile
Publish Web App	App deployed
Restart Web App	App restarted

OPERATION	DESCRIPTION
Start Web App	App started
Stop Web App	App stopped
Swap Web App Slots	Slots were swapped
Get Web App Slots Differences	Slot differences
Apply Web App Configuration	Applied configuration changes
Reset Web App Configuration	Configuration changes reset
Approve Private Endpoint Connections	Approved private endpoint connections
Network Trace Web Apps	Started network trace
Newpassword Web Apps	New password created
Get Zipped Container Logs for Web App	Get container logs
Restore Web App From Backup Blob	App restored from backup

For more information on the schema of Activity Log entries, see [Activity Log schema](#).

## See Also

- See [Monitoring Azure App Service](#) for a description of monitoring Azure App Service.
- See [Monitoring Azure resources with Azure Monitor](#) for details on monitoring Azure resources.

# Environment variables and app settings in Azure App Service

11/2/2021 • 32 minutes to read • [Edit Online](#)

In [Azure App Service](#), certain settings are available to the deployment or runtime environment as environment variables. Some of these settings can be customized when you set them manually as [app settings](#). This reference shows the variables you can use or customize.

## App environment

The following environment variables are related to the app environment in general.

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITE_SITE_NAME</code>	Read-only. App name.	
<code>WEBSITE_RESOURCE_GROUP</code>	Read-only. Azure resource group name that contains the app resource.	
<code>WEBSITE_OWNER_NAME</code>	Read-only. Contains the Azure subscription ID that owns the app, the resource group, and the webspace.	
<code>REGION_NAME</code>	Read-only. Region name of the app.	
<code>WEBSITE_PLATFORM_VERSION</code>	Read-only. App Service platform version.	
<code>HOME</code>	Read-only. Path to the home directory (for example, <code>D:\home</code> for Windows).	
<code>SERVER_PORT</code>	Read-only. The port the app should listen to.	
<code>WEBSITE_WARMUP_PATH</code>	A relative path to ping to warm up the app, beginning with a slash. The default is <code>/</code> , which pings the root path. The specific path can be pinged by an unauthenticated client, such as Azure Traffic Manager, even if <a href="#">App Service authentication</a> is set to reject unauthenticated clients. (NOTE: This app setting does not change the path used by AlwaysOn.)	
<code>WEBSITE_COMPUTE_MODE</code>	Read-only. Specifies whether app runs on dedicated ( <code>Dedicated</code> ) or shared ( <code>Shared</code> ) VM/s.	
<code>WEBSITE_SKU</code>	Read-only. SKU of the app. Possible values are <code>Free</code> , <code>Shared</code> , <code>Basic</code> , and <code>Standard</code> .	

SETTING NAME	DESCRIPTION	EXAMPLE
<code>SITE_BITNESS</code>	Read-only. Shows whether the app is 32-bit ( <code>x86</code> ) or 64-bit ( <code>AMD64</code> ).	
<code>WEBSITE_HOSTNAME</code>	Read-only. Primary hostname for the app. Custom hostnames are not accounted for here.	
<code>WEBSITE_VOLUME_TYPE</code>	Read-only. Shows the storage volume type currently in use.	
<code>WEBSITE_NPM_DEFAULT_VERSION</code>	Default npm version the app is using.	
<code>WEBSOCKET_CONCURRENT_REQUEST_LIMIT</code>	Read-only. Limit for websocket's concurrent requests. For <b>Standard</b> tier and above, the value is <code>-1</code> , but there's still a per VM limit based on your VM size (see <a href="#">Cross VM Numerical Limits</a> ).	
<code>WEBSITE_PRIVATE_EXTENSIONS</code>	Set to <code>0</code> to disable the use of private site extensions.	
<code>WEBSITE_TIME_ZONE</code>	By default, the time zone for the app is always UTC. You can change it to any of the valid values that are listed in <a href="#">TimeZone</a> . If the specified value isn't recognized, UTC is used.	<code>Atlantic Standard Time</code>
<code>WEBSITE_ADD_SITENAME_BINDINGS_IN_APPHOSTNAME</code>	On the <a href="#">hosting</a> of a storage volume failover or reconfiguration, your app is switched over to a standby storage volume. The default setting of <code>1</code> prevents your worker process from recycling when the storage infrastructure changes. If you are running a Windows Communication Foundation (WCF) app, disable it by setting it to <code>0</code> . The setting is slot-specific, so you should set it in all slots.	
<code>WEBSITE_PROACTIVE_AUTOHEAL_ENABLED</code>	By default, a VM instance is proactively "autohealed" when it's using more than 90% of allocated memory for more than 30 seconds, or when 80% of the total requests in the last two minutes take longer than 200 seconds. If a VM instance has triggered one of these rules, the recovery process is an overlapping restart of the instance. Set to <code>false</code> to disable this recovery behavior. The default is <code>true</code> . For more information, see <a href="#">Proactive Auto Heal</a> .	

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITE_PROACTIVE_CRASHMONITORING_ENABLED</code>	Whenever the w3wp.exe process on a VM instance of your app crashes due to an unhandled exception for more than three times in 24 hours, a debugger process is attached to the main worker process on that instance, and collects a memory dump when the worker process crashes again. This memory dump is then analyzed and the call stack of the thread that caused the crash is logged in your App Service's logs. Set to <code>false</code> to disable this automatic monitoring behavior. The default is <code>true</code> . For more information, see <a href="#">Proactive Crash Monitoring</a> .	
<code>WEBSITE_DAAS_STORAGE_SASURI</code>	During crash monitoring (proactive or manual), the memory dumps are deleted by default. To save the memory dumps to a storage blob container, specify the SAS URI.	
<code>WEBSITE_CRASHMONITORING_ENABLED</code>	Set to <code>true</code> to enable <a href="#">crash monitoring</a> manually. You must also set <code>WEBSITE_DAAS_STORAGE_SASURI</code> and <code>WEBSITE_CRASHMONITORING_SETTINGS</code> . The default is <code>false</code> . This setting has no effect if remote debugging is enabled. Also, if this setting is set to <code>true</code> , <a href="#">proactive crash monitoring</a> is disabled.	
<code>WEBSITE_CRASHMONITORING_SETTINGS</code>	A JSON with the following format: <pre>{"StartTimeUtc": "2020-02-10T08:21", "MaxHours": "&lt;elapsed-hours-from-StartTimeUtc&gt;", "MaxDumpCount": "&lt;max-number-of-crash-dumps&gt;"}</pre> . Required to configure <a href="#">crash monitoring</a> if <code>WEBSITE_CRASHMONITORING_ENABLED</code> is specified. To only log the call stack without saving the crash dump in the storage account, add <code>, "UseStorageAccount": "false"</code> in the JSON.	
<code>REMOTEDEBUGGINGVERSION</code>	Remote debugging version.	
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRINGS</code>	By default, App Service creates a shared storage for you at app creation. To use a custom storage account instead, set to the connection string of your storage account. For functions, see <a href="#">App settings reference for Functions</a> .	<code>DefaultEndpointsProtocol=https;AccountName=&lt;name&gt;;AccountKey=&lt;key&gt;</code>

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITE_CONTENTSHARE</code>	When you use specify a custom storage account with <code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code> , App Service creates a file share in that storage account for your app. To use a custom name, set this variable to the name you want. If a file share with the specified name doesn't exist, App Service creates it for you.	<code>myapp123</code>
<code>WEBSITE_SCM_ALWAYS_ON_ENABLED</code>	Read-only. Shows whether Always On is enabled ( <code>1</code> ) or not ( <code>0</code> ).	
<code>WEBSITE_SCM_SEPARATE_STATUS</code>	Read-only. Shows whether the Kudu app is running in a separate process ( <code>1</code> ) or not ( <code>0</code> ).	

## Variable prefixes

The following table shows environment variables prefixes that App Service uses for various purposes.

SETTING NAME	DESCRIPTION
<code>APPSETTING_</code>	Signifies that a variable is set by the customer as an app setting in the app configuration. It's injected into a .NET app as an app setting.
<code>MAINSITE_</code>	Signifies a variable is specific to the app itself.
<code>SCMSITE_</code>	Signifies a variable is specific to the Kudu app.
<code>SQLCONNSTR_</code>	Signifies a SQL Server connection string in the app configuration. It's injected into a .NET app as a connection string.
<code>SQLAZURECONNSTR_</code>	Signifies an Azure SQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
<code>POSTGRESQLCONNSTR_</code>	Signifies a PostgreSQL connection string in the app configuration. It's injected into a .NET app as a connection string.
<code>CUSTOMCONNSTR_</code>	Signifies a custom connection string in the app configuration. It's injected into a .NET app as a connection string.
<code>MYSQLCONNSTR_</code>	Signifies an Azure SQL Database connection string in the app configuration. It's injected into a .NET app as a connection string.
<code>AZUREFILESTORAGE_</code>	A connection string to a custom share for a container app in Azure Files.
<code>AZUREBLOBSTORAGE_</code>	A connection string to a custom storage account for a container app in Azure Blob Storage.

SETTING NAME	DESCRIPTION
NOTIFICATIONHUBCONNSTR_	Signifies a connection string to a notification hub in Azure Notification Hubs.
SERVICEBUSCONNSTR_	Signifies a connection string to an instance of Azure Service Bus.
EVENTHUBCONNSTR_	Signifies a connection string to an event hub in Azure Event Hubs.
DOCDBCONNSTR_	Signifies a connection string to a database in Azure Cosmos DB.
REDISCACHECONNSTR_	Signifies a connection string to a cache in Azure Cache for Redis.
FILESHARESTORAGE_	Signifies a connection string to a custom file share.

## Deployment

The following environment variables are related to app deployment. For variables related to App Service build automation, see [Build automation](#).

SETTING NAME	DESCRIPTION
DEPLOYMENT_BRANCH	For <a href="#">local Git</a> or <a href="#">cloud Git</a> deployment (such as GitHub), set to the branch in Azure you want to deploy to. By default, it is <code>master</code> .
WEBSITE_RUN_FROM_PACKAGE	Set to <code>1</code> to run the app from a local ZIP package, or set to the URL of an external URL to run the app from a remote ZIP package. For more information, see <a href="#">Run your app in Azure App Service directly from a ZIP package</a> .
WEBSITE_USE_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .
WEBSITE_RUN_FROM_ZIP	Deprecated. Use <code>WEBSITE_RUN_FROM_PACKAGE</code> .
WEBSITE_WEBDEPLOY_USE_SCM	Set to <code>false</code> for WebDeploy to stop using the Kudu deployment engine. The default is <code>true</code> . To deploy to Linux apps using Visual Studio (WebDeploy/MSDeploy), set it to <code>false</code> .
MSDEPLOY_RENAME_LOCKED_FILES	Set to <code>1</code> to attempt to rename DLLs if they can't be copied during a WebDeploy deployment. This setting is not applicable if <code>WEBSITE_WEBDEPLOY_USE_SCM</code> is set to <code>false</code> .
WEBSITE_DISABLE_SCM_SEPARATION	By default, the main app and the Kudu app run in different sandboxes. When you stop the app, the Kudu app is still running, and you can continue to use Git deploy and MSDeploy. Each app has its own local files. Turning off this separation (setting to <code>true</code> ) is a legacy mode that's no longer fully supported.

SETTING NAME	DESCRIPTION
<code>WEBSITE_ENABLE_SYNC_UPDATE_SITE</code>	Set to <code>1</code> ensure that REST API calls to update <code>site</code> and <code>siteconfig</code> are completely applied to all instances before returning. The default is <code>1</code> if deploying with an ARM template, to avoid race conditions with subsequent ARM calls.
<code>WEBSITE_START_SCM_ON_SITE_CREATION</code>	In an ARM template deployment, set to <code>1</code> in the ARM template to pre-start the Kudu app as part of app creation.
<code>WEBSITE_START_SCM_WITH_PRELOAD</code>	For Linux apps, set to <code>true</code> to force preloading the Kudu app when Always On is enabled by pinging its URL. The default is <code>false</code> . For Windows apps, the Kudu app is always preloaded.

## Build automation

- [Kudu \(Windows\)](#)
- [Oryx \(Linux\)](#)

Kudu build configuration applies to native Windows apps and is used to control the behavior of Git-based (or ZIP-based) deployments.

SETTING NAME	DESCRIPTION	EXAMPLE
<code>SCM_BUILD_ARGS</code>	Add things at the end of the msbuild command line, such that it overrides any previous parts of the default command line.	To do a clean build: <code>-t:Clean;Compile</code>
<code>SCM_SCRIPT_GENERATOR_ARGS</code>	Kudu uses the <code>azure site deploymentscript</code> command described <a href="#">here</a> to generate a deployment script. It automatically detects the language framework type and determines the parameters to pass to the command. This setting overrides the automatically generated parameters.	To treat your repository as plain content files: <code>--basic -p &lt;folder-to-deploy&gt;</code>
<code>SCM_TRACE_LEVEL</code>	Build trace level. The default is <code>1</code> . Set to higher values, up to <code>4</code> , for more tracing.	<code>4</code>
<code>SCM_COMMAND_IDLE_TIMEOUT</code>	Time-out in seconds for each command that the build process launches to wait before without producing any output. After that, the command is considered idle and killed. The default is <code>60</code> (one minute). In Azure, there's also a general idle request timeout that disconnects clients after 230 seconds. However, the command will still continue running server-side after that.	
<code>SCM_LOGSTREAM_TIMEOUT</code>	Time-out of inactivity in seconds before stopping log streaming. The default is <code>1800</code> (30 minutes).	

SETTING NAME	DESCRIPTION	EXAMPLE
<code>SCM_SITEEXTENSIONS_FEED_URL</code>	URL of the site extensions gallery. The default is <code>https://www.nuget.org/api/v2/</code> . The URL of the old feed is <code>http://www.siteextensions.net/api/v2/</code>	
<code>SCM_USE_LIBGIT2SHARP_REPOSITORY</code>	Set to <code>0</code> to use git.exe instead of libgit2sharp for git operations.	
<code>WEBSITE_LOAD_USER_PROFILE</code>	In case of the error <code>The specified user does not have a valid profile.</code> during ASP.NET build automation (such as during Git deployment), set this variable to <code>1</code> to load a full user profile in the build environment. This setting is only applicable when <code>WEBSITE_COMPUTE_MODE</code> is <code>Dedicated</code> .	
<code>WEBSITE_SCM_IDLE_TIMEOUT_IN_MINUTES</code>	Time-out in minutes for the SCM (Kudu) site. The default is <code>20</code> .	
<code>SCM_DO_BUILD_DURING_DEPLOYMENT</code>	With <a href="#">ZIP deploy</a> , the deployment engine assumes that a ZIP file is ready to run as-is and doesn't run any build automation. To enable the same build automation as in <a href="#">Git deploy</a> , set to <code>true</code> .	

## Language-specific settings

This section shows the configurable runtime settings for each supported language framework. Additional settings are available during [build automation](#) at deployment time.

- [.NET](#)
- [Java](#)
- [Node.js](#)
- [Python](#)
- [PHP](#)
- [Ruby](#)

SETTING NAME	DESCRIPTION
<code>PORT</code>	Read-only. For Linux apps, port that the .NET runtime listens to in the container.
<code>WEBSITE_ROLE_INSTANCE_ID</code>	Read-only. ID of the current instance.
<code>HOME</code>	Read-only. Directory that points to shared storage ( <code>/home</code> ).
<code>DUMP_DIR</code>	Read-only. Directory for the crash dumps ( <code>/home/logs/dumps</code> ).

SETTING NAME	DESCRIPTION
APP_SVC_RUN_FROM_COPY	Linux apps only. By default, the app is run from <code>/home/site/wwwroot</code> , a shared directory for all scaled-out instances. Set this variable to <code>true</code> to copy the app to a local directory in your container and run it from there. When using this option, be sure not to hard-code any reference to <code>/home/site/wwwroot</code> . Instead, use a path relative to <code>/home/site/wwwroot</code> .
MACHINEKEY_Decryption	For Windows native apps or Windows container apps, this variable is injected into app environment or container to enable ASP.NET cryptographic routines (see <a href="#">machineKey Element</a> ). To override the default <code>decryption</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <i>Web.config</i> file.
MACHINEKEY_DecryptionKey	For Windows native apps or Windows container apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see <a href="#">machineKey Element</a> ). To override the automatically generated <code>decryptionKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <i>Web.config</i> file.
MACHINEKEY_Validation	For Windows native apps or Windows container apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see <a href="#">machineKey Element</a> ). To override the default <code>validation</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <i>Web.config</i> file.
MACHINEKEY_ValidationKey	For Windows native apps or Windows container apps, this variable is injected into the app environment or container to enable ASP.NET cryptographic routines (see <a href="#">machineKey Element</a> ). To override the automatically generated <code>validationKey</code> value, configure it as an App Service app setting, or set it directly in the <code>machineKey</code> element of the <i>Web.config</i> file.

## Domain and DNS

SETTING NAME	DESCRIPTION	EXAMPLE
WEBSITE_DNS_SERVER	IP address of primary DNS server for outgoing connections (such as to a back-end service). The default DNS server for App Service is Azure DNS, whose IP address is <code>168.63.129.16</code> . If your app uses <a href="#">VNet integration</a> or is in an <a href="#">App Service environment</a> , it inherits the DNS server configuration from the VNet by default.	<code>10.0.0.1</code>
WEBSITE_DNS_ALT_SERVER	IP address of fallback DNS server for outgoing connections. See <code>WEBSITE_DNS_SERVER</code> .	

## TLS/SSL

For more information, see [Use a TLS/SSL certificate in your code in Azure App Service](#).

SETTING NAME	DESCRIPTION
WEBSITE_LOAD_CERTIFICATES	Comma-separate thumbprint values to the certificate you want to load in your code, or <code>*</code> to allow all certificates to be loaded in code. Only <a href="#">certificates added to your app</a> can be loaded.
WEBSITE_PRIVATE_CERTS_PATH	Read-only. Path in a Windows container to the loaded private certificates.
WEBSITE_PUBLIC_CERTS_PATH	Read-only. Path in a Windows container to the loaded public certificates.
WEBSITE_INTERMEDIATE_CERTS_PATH	Read-only. Path in a Windows container to the loaded intermediate certificates.
WEBSITE_ROOT_CERTS_PATH	Read-only. Path in a Windows container to the loaded root certificates.

## Deployment slots

For more information on deployment slots, see [Set up staging environments in Azure App Service](#).

SETTING NAME	DESCRIPTION	EXAMPLE
WEBSITE_SLOT_NAME	Read-only. Name of the current deployment slot. The name of the production slot is <code>Production</code> .	
WEBSITE_OVERRIDE_STICKY_EXTENSION_VERSIONS	By default, the versions for site extensions are specific to each slot. This prevents unanticipated application behavior due to changing extension versions after a swap. If you want the extension versions to swap as well, set to <code>1</code> on <i>all slots</i> .	
WEBSITE_OVERRIDE_PRESERVE_DEFAULT_STICKY_SETTINGS	Designates certain settings as <b>sticky</b> or <b>not swappable by default</b> . Default is <code>true</code> . Set this setting to <code>false</code> or <code>0</code> for <i>all deployment slots</i> to make them swappable instead. There's no fine-grain control for specific setting types.	
WEBSITE_SWAP_WARMUP_PING_PATH	Path to ping to warm up the target slot in a swap, beginning with a slash. The default is <code>/</code> , which pings the root path.	<code>/statuscheck</code>
WEBSITE_SWAP_WARMUP_PING_STATUSES	Valid HTTP response codes for the warm-up operation during a swap. If the returned status code isn't in the list, the warmup and swap operations are stopped. By default, all response codes are valid.	<code>200,202</code>

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITE_SLOT_NUMBER_OF_TIMEOUTS_BEFORE_SWAP</code>	During a slot swap, maximum number of timeouts after which we force restart the site on a specific VM instance. The default is <code>3</code> .	
<code>WEBSITE_SLOT_MAX_NUMBER_OF_TIMEOUTS</code>	During a slot swap, maximum number of timeout requests for a single URL to make before giving up. The default is <code>5</code> .	
<code>WEBSITE_SKIP_ALL_BINDINGS_IN_APPHOST</code>	Set to <code>true</code> or <code>1</code> to skip all bindings in <code>applicationHost.config</code> . The default is <code>false</code> . If your app triggers a restart because <code>applicationHost.config</code> is updated with the swapped hostnames of the slots, set this variable to <code>true</code> to avoid a restart of this kind. If you are running a Windows Communication Foundation (WCF) app, do not set this variable.	

## Custom containers

For more information on custom containers, see [Run a custom container in Azure](#).

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code>	Set to <code>true</code> to enable the <code>/home</code> directory to be shared across scaled instances. The default is <code>false</code> for custom containers.	
<code>WEBSITES_CONTAINER_START_TIME_LIMIT</code>	Amount of time in seconds to wait for the container to complete start-up before restarting the container. Default is <code>230</code> . You can increase it up to the maximum of <code>1800</code> .	
<code>DOCKER_REGISTRY_SERVER_URL</code>	URL of the registry server, when running a custom container in App Service. For security, this variable is not passed on to the container.	<code>https://&lt;server-name&gt;.azurecr.io</code>
<code>DOCKER_REGISTRY_SERVER_USERNAME</code>	Username to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable is not passed on to the container.	
<code>DOCKER_REGISTRY_SERVER_PASSWORD</code>	Password to authenticate with the registry server at <code>DOCKER_REGISTRY_SERVER_URL</code> . For security, this variable is not passed on to the container.	

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITES_WEB_CONTAINER_NAME</code>	In a Docker Compose app, only one of the containers can be internet accessible. Set to the name of the container defined in the configuration file to override the default container selection. By default, the internet accessible container is the first container to define port 80 or 8080, or, when no such container is found, the first container defined in the configuration file.	
<code>WEBSITES_PORT</code>	For a custom container, the custom port number on the container for App Service to route requests to. By default, App Service attempts automatic port detection of ports 80 and 8080. This setting is <i>not</i> injected into the container as an environment variable.	
<code>WEBSITE_CPU_CORES_LIMIT</code>	By default, a Windows container runs with all available cores for your chosen pricing tier. To reduce the number of cores, set to the number of desired cores limit. For more information, see <a href="#">Customize the number of compute cores</a> .	
<code>WEBSITE_MEMORY_LIMIT_MB</code>	By default all Windows Containers deployed in Azure App Service are limited to 1 GB RAM. Set to the desired memory limit in MB. The cumulative total of this setting across apps in the same plan must not exceed the amount allowed by the chosen pricing tier. For more information, see <a href="#">Customize container memory</a> .	
<code>CONTAINER_WINRM_ENABLED</code>	For a Windows container app, set to <code>1</code> to enable Windows Remote Management (WIN-RM).	

## Scaling

SETTING NAME	DESCRIPTION
<code>WEBSITE_INSTANCE_ID</code>	Read-only. Unique ID of the current VM instance, when the app is scaled out to multiple instances.
<code>WEBSITE_IIS_SITE_NAME</code>	Deprecated. Use <code>WEBSITE_INSTANCE_ID</code> .
<code>WEBSITE_DISABLE_OVERLAPPED_RECYCLING</code>	Overlapped recycling makes it so that before the current VM instance of an app is shut down, a new VM instance starts. In some cases, it can cause file locking issues. You can try turning it off by setting to <code>1</code> .

SETTING NAME	DESCRIPTION
<code>WEBSITE_DISABLE_CROSS_STAMP_SCALE</code>	By default, apps are allowed to scale across stamps if they use Azure Files or a Docker container. Set to <code>1</code> or <code>true</code> to disable cross-stamp scaling within the app's region. The default is <code>0</code> . Custom Docker containers that set <code>WEBSITES_ENABLE_APP_SERVICE_STORAGE</code> to <code>true</code> or <code>1</code> cannot scale cross-stamps because their content is not completely encapsulated in the Docker container.

## Logging

SETTING NAME	DESCRIPTION	EXAMPLE
<code>WEBSITE_HTTPLOGGING_ENABLED</code>	Read-only. Shows whether the web server logging for Windows native apps is enabled ( <code>1</code> ) or not ( <code>0</code> ).	
<code>WEBSITE_HTTPLOGGING_RETENTION_DAYS</code>	Retention period in days of web server logs for Windows native apps, if web server logs are enabled.	<code>10</code>
<code>WEBSITE_HTTPLOGGING_CONTAINER_URL</code>	SAS URL of the blob storage container to store web server logs for Windows native apps, if web server logs are enabled. If not set, web server logs are stored in the app's file system (default shared storage).	
<code>DIAGNOSTICS_AZUREBLOBRETENTIONINDAYS</code>	Retention period in days of application logs for Windows native apps, if application logs are enabled.	<code>10</code>
<code>DIAGNOSTICS_AZUREBLOBCONTAINERSASURL</code>	SAS URL of the blob storage container to store application logs for Windows native apps, if application logs are enabled.	
<code>APPSERVICEAPPLOGS_TRACE_LEVEL</code>	Minimum log level to ship to Log Analytics for the <a href="#">AppServiceAppLogs</a> log type.	
<code>DIAGNOSTICS_LASTRESORTFILE</code>	The filename to create, or a relative path to the log directory, for logging internal errors for troubleshooting the listener. The default is <code>logging-errors.txt</code> .	
<code>DIAGNOSTICS_LOGGINGSETTINGSFILE</code>	Path to the log settings file, relative to <code>D:\home</code> or <code>/home</code> . The default is <code>site\diagnostics\settings.json</code> .	
<code>DIAGNOSTICS_TEXTTRACELOGDIRECTORY</code>	The log folder, relative to the app root ( <code>D:\home\site\wwwroot</code> or <code>/home/site/wwwroot</code> ).	<code>...\\LogFiles\\Application</code>
<code>DIAGNOSTICS_TEXTTRACEMAXLOGFILESIZEBY</code>	Maximum size of the log file in bytes. The default is <code>131072</code> (128 KB).	

SETTING NAME	DESCRIPTION	EXAMPLE
DIAGNOSTICS_TEXTTRACE_MAX_LOG_FOLDER_SIZE	Maximum size of the log folder in bytes. The default is <code>1048576</code> (1 MB).	
DIAGNOSTICS_TEXTTRACE_MAX_NUM_LOGFILES	Maximum number of log files to keep. The default is <code>20</code> .	
DIAGNOSTICS_TEXTTRACE_TURNOFFPERIOD	Time-out in milliseconds to keep application logging enabled. The default is <code>43200000</code> (12 hours).	
WEBSITE_LOG_BUFFERING	By default, log buffering is enabled. Set to <code>0</code> to disable it.	
WEBSITE_ENABLE_PERF_MODE	For native Windows apps, set to <code>TRUE</code> to turn off IIS log entries for successful requests returned within 10 seconds. This is a quick way to do performance benchmarking by removing extended logging.	

## Performance counters

The following are 'fake' environment variables that don't exist if you enumerate them, but return their value if you look them up individually. The value is dynamic and can change on every lookup.

SETTING NAME	DESCRIPTION
WEBSITE_COUNTERS_ASPNET	A JSON object containing the ASP.NET perf counters.
WEBSITE_COUNTERS_APP	A JSON object containing sandbox counters.
WEBSITE_COUNTERS_CLR	A JSON object containing CLR counters.
WEBSITE_COUNTERS_ALL	A JSON object containing the combination of the other three variables.

## Caching

SETTING NAME	DESCRIPTION
WEBSITE_LOCAL_CACHE_OPTION	Whether local cache is enabled. Available options are: <ul style="list-style-type: none"> <li>- <code>Default</code> : Inherit the stamp-level global setting.</li> <li>- <code>Always</code> : Enable for the app.</li> <li>- <code>OnStorageUnavailability</code></li> <li>- <code>Disabled</code> : Disabled for the app.</li> </ul>
WEBSITE_LOCAL_CACHE_READWRITE_OPTION	Read-write options of the local cache. Available options are: <ul style="list-style-type: none"> <li>- <code>ReadOnly</code> : Cache is read-only.</li> <li>- <code>WriteWithCopyBack</code> : Allow writes to local cache and copy periodically to shared storage. Applicable only for single instance apps as the SCM site points to local cache.</li> <li>- <code>WriteButDiscardChanges</code> : Allow writes to local cache but discard changes made locally.</li> </ul>
WEBSITE_LOCAL_CACHE_SIZEINMB	Size of the local cache in MB. Default is <code>1000</code> (1 GB).

SETTING NAME	DESCRIPTION
WEBSITE_LOCALCACHE_READY	Read-only flag indicating if the app using local cache.
WEBSITE_DYNAMIC_CACHE	Due to network file shared nature to allow access for multiple instances, the dynamic cache improves performance by caching the recently accessed files locally on an instance. Cache is invalidated when file is modified. The cache location is <code>%SYSTEMDRIVE%\local\DynamicCache</code> (same <code>%SYSTEMDRIVE%\local</code> quota is applied). By default, full content caching is enabled (set to <code>1</code> ), which includes both file content and directory/file metadata (timestamps, size, directory content). To conserve local disk use, set to <code>2</code> to cache only directory/file metadata (timestamps, size, directory content). To turn off caching, set to <code>0</code> .
WEBSITE_READONLY_APP	When using dynamic cache, you can disable write access to the app root ( <code>D:\home\site\wwwroot</code> or <code>/home/site/wwwroot</code> ) by setting this variable to <code>1</code> . Except for the <code>App_Data</code> directory, no exclusive locks are allowed, so that deployments don't get blocked by locked files.

## Networking

The following environment variables are related to [hybrid connections](#) and [VNET integration](#).

SETTING NAME	DESCRIPTION
WEBSITE_RELAYS	Read-only. Data needed to configure the Hybrid Connection, including endpoints and service bus data.
WEBSITE_REWRITE_TABLE	Read-only. Used at runtime to do the lookups and rewrite connections appropriately.
WEBSITE_VNET_ROUTE_ALL	By default, if you use <a href="#">regional VNet Integration</a> , your app only routes RFC1918 traffic into your VNet. Set to <code>1</code> to route all outbound traffic into your VNet and be subject to the same NSGs and UDRs. The setting lets you access non-RFC1918 endpoints through your VNet, secure all outbound traffic leaving your app, and force tunnel all outbound traffic to a network appliance of your own choosing.
WEBSITE_PRIVATE_IP	Read-only. IP address associated with the app when <a href="#">integrated with a VNet</a> . For Regional VNet Integration, the value is an IP from the address range of the delegated subnet, and for Gateway-required VNet Integration, the value is an IP from the address range of the point-to-site address pool configured on the Virtual Network Gateway. This IP is used by the app to connect to the resources through the VNet. Also, it can change within the described address range.
WEBSITE_PRIVATE_PORTS	Read-only. In VNet integration, shows which ports are useable by the app to communicate with other nodes.

## Key vault references

The following environment variables are related to [key vault references](#).

SETTING NAME	DESCRIPTION
<code>WEBSITE_KEYVAULT_REFERENCES</code>	Read-only. Contains information (including statuses) for all Key Vault references that are currently configured in the app.
<code>WEBSITE_SKIP_CONTENTSHARE_VALIDATION</code>	If you set the shared storage connection of your app (using <code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code> ) to a Key Vault reference, the app cannot resolve the key vault reference at app creation or update if one of the following conditions is true: <ul style="list-style-type: none"> <li>- The app accesses the key vault with a system-assigned identity.</li> <li>- The app accesses the key vault with a user-assigned identity, and the key vault is <a href="#">locked with a VNet</a>.</li> </ul> To avoid errors at create or update time, set this variable to <code>1</code> .
<code>WEBSITE_DELAY_CERT_DELETION</code>	This env var can be set to 1 by users in order to ensure that a certificate that a worker process is dependent upon is not deleted until it exits.

## CORS

The following environment variables are related to Cross-Origin Resource Sharing (CORS) configuration.

SETTING NAME	DESCRIPTION
<code>WEBSITE_CORS_ALLOWED_ORIGINS</code>	Read-only. Shows the allowed origins for CORS.
<code>WEBSITE_CORS_SUPPORT_CREDENTIALS</code>	Read-only. Shows whether setting the <code>Access-Control-Allow-Credentials</code> header to <code>true</code> is enabled ( <code>True</code> ) or not ( <code>False</code> ).

## Authentication & Authorization

The following environment variables are related to [App Service authentication](#).

SETTING NAME	DESCRIPTION
<code>WEBSITE_AUTH_DISABLE_IDENTITY_FLOW</code>	When set to <code>true</code> , disables assigning the thread principal identity in ASP.NET-based web applications (including v1 Function Apps). This is designed to allow developers to protect access to their site with auth, but still have it use a separate login mechanism within their app logic. The default is <code>false</code> .
<code>WEBSITE_AUTH_HIDE_DEPRECATED_SID</code>	<code>true</code> or <code>false</code> . The default value is <code>false</code> . This is a setting for the legacy Azure Mobile Apps integration for Azure App Service. Setting this to <code>true</code> resolves an issue where the SID (security ID) generated for authenticated users might change if the user changes their profile information. Changing this value may result in existing Azure Mobile Apps user IDs changing. Most apps do not need to use this setting.

SETTING NAME	DESCRIPTION
WEBSITE_AUTH_NONCE_DURATION	A <i>timespan</i> value in the form <code>_hours_:_minutes_:_seconds_</code> . The default value is <code>00:05:00</code> , or 5 minutes. This setting controls the lifetime of the <a href="#">cryptographic nonce</a> generated for all browser-driven logins. If a login fails to complete in the specified time, the login flow will be retried automatically. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.nonceExpirationInterval</code> configuration value.
WEBSITE_AUTH_PRESERVE_URL_FRAGMENT	When set to <code>true</code> and users click on app links that contain URL fragments, the login process will ensure that the URL fragment part of your URL does not get lost in the login redirect process. For more information, see <a href="#">Customize sign-in and sign-out in Azure App Service authentication</a> .
WEBSITE_AUTH_USE_LEGACY CLAIMS	To maintain backward compatibility across upgrades, the authentication module uses the legacy claims mapping of short to long names in the <code>/auth/me</code> API, so certain mappings are excluded (e.g. "roles"). To get the more modern version of the claims mappings, set this variable to <code>False</code> . In the "roles" example, it would be mapped to the long claim name " <a href="http://schemas.microsoft.com/ws/2008/06/identity/claims/role">http://schemas.microsoft.com/ws/2008/06/identity/claims/role</a> ".
WEBSITE_AUTH_DISABLE_WWWAUTHENTICATE	<code>true</code> or <code>false</code> . The default value is <code>false</code> . When set to <code>true</code> , removes the <code>WWW-Authenticate</code> HTTP response header from module-generated HTTP 401 responses. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>identityProviders.azureActiveDirectory.login.disableWwwAuthenticate</code> configuration value.
WEBSITE_AUTH_STATE_DIRECTORY	A local file system directory path where tokens are stored when the file-based token store is enabled. The default value is <code>%HOME%\Data\auth</code> . This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.FileSystem.directory</code> configuration value.
WEBSITE_AUTH_TOKEN_CONTAINER_SASURL	A fully qualified blob container URL. Instructs the auth module to store and load all encrypted tokens to the specified blob storage container instead of using the default local file system.
WEBSITE_AUTH_TOKEN_REFRESH_HOURS	Any positive decimal number. The default value is <code>72</code> (hours). This setting controls the amount of time after a session token expires that the <code>/auth/refresh</code> API can be used to refresh it. It's intended primarily for use with Azure Mobile Apps, which rely on session tokens. Refresh attempts after this period will fail and end users will be required to sign-in again. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.tokenStore.tokenRefreshExtensionHours</code> configuration value.

SETTING NAME	DESCRIPTION
WEBSITE_AUTH_TRACE_LEVEL	Controls the verbosity of authentication traces written to <a href="#">Application Logging</a> . Valid values are <code>Off</code> , <code>Error</code> , <code>Warning</code> , <code>Information</code> , and <code>Verbose</code> . The default value is <code>Verbose</code> .
WEBSITE_AUTH_VALIDATE_NONCE	<code>true</code> or <code>false</code> . The default value is <code>true</code> . This value should never be set to <code>false</code> except when temporarily debugging <a href="#">cryptographic nonce</a> validation failures that occur during interactive logins. This application setting is intended for use with the V1 (classic) configuration experience. If using the V2 authentication configuration schema, you should instead use the <code>login.nonce.validateNonce</code> configuration value.
WEBSITE_AUTH_V2_CONFIG_JSON	This environment variable is populated automatically by the Azure App Service platform and is used to configure the integrated authentication module. The value of this environment variable corresponds to the V2 (non-classic) authentication configuration for the current app in Azure Resource Manager. It's not intended to be configured explicitly.
WEBSITE_AUTH_ENABLED	Read-only. Injected into a Windows or Linux app to indicate whether App Service authentication is enabled.
WEBSITE_AUTH_ENCRYPTION_KEY	By default, the automatically generated key is used as the encryption key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supercedes the <code>MACHINEKEY_DecryptionKey</code> setting.
WEBSITE_AUTH_SIGNING_KEY	By default, the automatically generated key is used as the signing key. To override, set to a desired key. This is recommended if you want to share tokens or sessions across multiple apps. If specified, it supercedes the <code>MACHINEKEY_ValidationKey</code> setting.

## Managed identity

The following environment variables are related to [managed identities](#).

SETTING NAME	DESCRIPTION
IDENTITY_ENDPOINT	Read-only. The URL to retrieve the token for the app's <a href="#">managed identity</a> .
MSI_ENDPOINT	Deprecated. Use <code>IDENTITY_ENDPOINT</code> .
IDENTITY_HEADER	Read-only. Value that must be added to the <code>X-IDENTITY-HEADER</code> header when making an HTTP GET request to <code>IDENTITY_ENDPOINT</code> . The value is rotated by the platform.
MSI_SECRET	Deprecated. Use <code>IDENTITY_HEADER</code> .

## Health check

The following environment variables are related to [health checks](#).

SETTING NAME	DESCRIPTION
WEBSITE_HEALTHCHECK_MAXPINGFAILURES	The maximum number of failed pings before removing the instance. Set to a value between <code>2</code> and <code>100</code> . When you are scaling up or out, App Service pings the Health check path to ensure new instances are ready. For more information, see <a href="#">Health check</a> .
WEBSITE_HEALTHCHECK_MAXUNHEALTHYWORKERPERCENT	To avoid overwhelming healthy instances, no more than half of the instances will be excluded. For example, if an App Service Plan is scaled to four instances and three are unhealthy, at most two will be excluded. The other two instances (one healthy and one unhealthy) will continue to receive requests. In the worst-case scenario where all instances are unhealthy, none will be excluded. To override this behavior, set to a value between <code>0</code> and <code>100</code> . A higher value means more unhealthy instances will be removed. The default is <code>50</code> (50%).

## Push notifications

The following environment variables are related to the [push notifications](#) feature.

SETTING NAME	DESCRIPTION
WEBSITE_PUSH_ENABLED	Read-only. Added when push notifications are enabled.
WEBSITE_PUSH_TAG_WHITELIST	Read-only. Contains the tags in the notification registration.
WEBSITE_PUSH_TAGS_REQUIRING_AUTH	Read-only. Contains a list of tags in the notification registration that requires user authentication.
WEBSITE_PUSH_TAGS_DYNAMIC	Read-only. Contains a list of tags in the notification registration that were added automatically.

### NOTE

This article contains references to the term *whitelist*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

## Webjobs

The following environment variables are related to [WebJobs](#).

SETTING NAME	DESCRIPTION
WEBJOBS_RESTART_TIME	For continuous jobs, delay in seconds when a job's process goes down for any reason before relaunching it.
WEBJOBS_IDLE_TIMEOUT	For triggered jobs, timeout in seconds, after which the job is aborted if it's in idle, has no CPU time or output.
WEBJOBS_HISTORY_SIZE	For triggered jobs, maximum number of runs kept in the history directory per job. The default is <code>50</code> .

SETTING NAME	DESCRIPTION
<code>WEBJOBS_STOPPED</code>	Set to <code>1</code> to disable running any job, and stop all currently running jobs.
<code>WEBJOBS_DISABLE_SCHEDULE</code>	Set to <code>1</code> to turn off all scheduled triggering. Jobs can still be manually invoked.
<code>WEBJOBS_ROOT_PATH</code>	Absolute or relative path of webjob files. For a relative path, the value is combined with the default root path ( <code>D:/home/site/wwwroot/</code> or <code>/home/site/wwwroot/</code> ).
<code>WEBJOBS_LOG_TRIGGERED_JOBS_TO_APP_LOGS</code>	Set to true to send output from triggered WebJobs to the application logs pipeline (which supports file system, blobs, and tables).
<code>WEBJOBS_SHUTDOWN_FILE</code>	File that App Service creates when a shutdown request is detected. It's the web job process's responsibility to detect the presence of this file and initiate shutdown. When using the WebJobs SDK, this part is handled automatically.
<code>WEBJOBS_PATH</code>	Read-only. Root path of currently running job (will be under some temporary directory).
<code>WEBJOBS_NAME</code>	Read-only. Current job name.
<code>WEBJOBS_TYPE</code>	Read-only. Current job type ( <code>triggered</code> or <code>continuous</code> ).
<code>WEBJOBS_DATA_PATH</code>	Read-only. Current job metadata path to contain the job's logs, history, and any artifact of the job.
<code>WEBJOBS_RUN_ID</code>	Read-only. For triggered jobs, current run ID of the job.

## Functions

SETTING NAME	DESCRIPTION
<code>WEBSITE_FUNCTIONS_ARMCACHE_ENABLED</code>	Set to <code>0</code> to disable the functions cache.
<code>WEBSITE_MAX_DYNAMIC_APPLICATION_SCALE_OUT</code>	<a href="#">App settings reference for Azure Functions</a>
<code>FUNCTIONS_EXTENSION_VERSION</code>	<a href="#">App settings reference for Azure Functions</a>
<code>AzureWebJobsSecretStorageType</code>	<a href="#">App settings reference for Azure Functions</a>
<code>FUNCTIONS_WORKER_RUNTIME</code>	<a href="#">App settings reference for Azure Functions</a>
<code>AzureWebJobsStorage</code>	<a href="#">App settings reference for Azure Functions</a>
<code>WEBSITE_CONTENTAZUREFILECONNECTIONSTRING</code>	<a href="#">App settings reference for Azure Functions</a>
<code>WEBSITE_CONTENTSHARE</code>	<a href="#">App settings reference for Azure Functions</a>
<code>WEBSITE_CONTENTOVERVNET</code>	<a href="#">App settings reference for Azure Functions</a>

SETTING NAME	DESCRIPTION
WEBSITE_ENABLE_BROTLI_ENCODING	<a href="#">App settings reference for Azure Functions</a>
WEBSITE_USE_PLACEHOLDER	Set to <code>0</code> to disable the placeholder functions optimization on the consumption plan. The placeholder is an optimization that <a href="#">improves the cold start</a> .
WEBSITE_PLACEHOLDER_MODE	Read-only. Shows whether the function app is running on a placeholder host ( <code>generalized</code> ) or its own host ( <code>specialized</code> ).
WEBSITE_DISABLE_ZIP_CACHE	When your app runs from a <a href="#">ZIP package</a> ( <code>WEBSITE_RUN_FROM_PACKAGE=1</code> ), the five most recently deployed ZIP packages are cached in the app's file system (D:\home\data\SitePackages). Set this variable to <code>1</code> to disable this cache. For Linux consumption apps, the ZIP package cache is disabled by default.

# Azure Policy built-in definitions for Azure App Service

11/2/2021 • 13 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy built-in policy definitions](#) for Azure App Service. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Azure App Service

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">API App should only be accessible over HTTPS</a>	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled	1.0.0
<a href="#">API apps should use an Azure file share for its content directory</a>	The content directory of an API app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to <a href="https://go.microsoft.com/fwlink/?linkid=2151594">https://go.microsoft.com/fwlink/?linkid=2151594</a> .	Audit, Disabled	1.0.0
<a href="#">App Service Apps should be injected into a virtual network</a>	Injecting App Service Apps in a virtual network unlocks advanced App Service networking and security features and provides you with greater control over your network security configuration. Learn more at: <a href="https://docs.microsoft.com/azure/app-service/web-sites-integrate-with-vnet">https://docs.microsoft.com/azure/app-service/web-sites-integrate-with-vnet</a> .	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">App Service apps should enable outbound non-RFC 1918 traffic to Azure Virtual Network</a>	By default, if one uses regional Azure Virtual Network (VNET) integration, the app only routes RFC1918 traffic into that respective virtual network. Using the API to set 'vnetRouteAllEnabled' to true enables all outbound traffic into the Azure Virtual Network. This setting allows features like network security groups and user defined routes to be used for all outbound traffic from the App Service app.	AuditIfNotExists, Disabled	1.0.0
<a href="#">App Service apps should use a SKU that supports private link</a>	With supported SKUs, Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to apps, you can reduce data leakage risks. Learn more about private links at: <a href="https://aka.ms/private-link">https://aka.ms/private-link</a> .	Audit, Deny, Disabled	1.0.0
<a href="#">App Service Environment apps should not be reachable over public internet</a>	To ensure apps deployed in an App Service Environment are not accessible over public internet, one should deploy App Service Environment with an IP address in virtual network. To set the IP address to a virtual network IP, the App Service Environment must be deployed with an internal load balancer.	Audit, Deny, Disabled	1.0.0
<a href="#">App Service Environment should be configured with strongest TLS Cipher suites</a>	The two most minimal and strongest cipher suites required for App Service Environment to function correctly are : TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 and TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.	Audit, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">App Service Environment should be provisioned with latest versions</a>	Only allow App Service Environment version 2 or version 3 to be provisioned. Older versions of App Service Environment require manual management of Azure resources and have greater scaling limitations.	Audit, Deny, Disabled	1.0.0
<a href="#">App Service Environment should disable TLS 1.0 and 1.1</a>	TLS 1.0 and 1.1 are out-of-date protocols that do not support modern cryptographic algorithms. Disabling inbound TLS 1.0 and 1.1 traffic helps secure apps in an App Service Environment.	Audit, Deny, Disabled	2.0.0
<a href="#">App Service Environment should enable internal encryption</a>	Setting InternalEncryption to true encrypts the pagefile, worker disks, and internal network traffic between the front ends and workers in an App Service Environment. To learn more, refer to <a href="https://docs.microsoft.com/azure/app-service/environment/app-service-app-service-environment-custom-settings#enable-internal-encryption">https://docs.microsoft.com/azure/app-service/environment/app-service-app-service-environment-custom-settings#enable-internal-encryption</a> .	Audit, Disabled	1.0.0
<a href="#">App Service should have local authentication methods disabled for FTP deployments</a>	Disabling local authentication methods improves security by ensuring that App Service exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	AuditIfNotExists, Disabled	1.0.0
<a href="#">App Service should have local authentication methods disabled for SCM site deployments</a>	Disabling local authentication methods improves security by ensuring that App Service exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	AuditIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">App Service should use a virtual network service endpoint</a>	This policy audits any App Service not configured to use a virtual network service endpoint.	AuditIfNotExists, Disabled	1.0.0
<a href="#">App Service should use private link</a>	Azure Private Link lets you connect your virtual networks to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to App Service, you can reduce data leakage risks. Learn more about private links at: <a href="https://aka.ms/private-link">https://aka.ms/private-link</a> .	AuditIfNotExists, Disabled	1.0.0
<a href="#">App Service slots should have local authentication methods disabled for FTP deployments</a>	Disabling local authentication methods improves security by ensuring that App Service slots exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	AuditIfNotExists, Disabled	1.0.0
<a href="#">App Service slots should have local authentication methods disabled for SCM site deployments</a>	Disabling local authentication methods improves security by ensuring that App Service slots exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	AuditIfNotExists, Disabled	1.0.0
<a href="#">App Services should disable public network access</a>	Disabling public network access improves security by ensuring that the App Service is not exposed on the public internet. Creating private endpoints can limit exposure of an App Service. Learn more at: <a href="https://aka.ms/app-service-private-endpoint">https://aka.ms/app-service-private-endpoint</a> .	AuditIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Authentication should be enabled on your API app</a>	Azure App Service Authentication is a feature that can prevent anonymous HTTP requests from reaching the API app, or authenticate those that have tokens before they reach the API app	AuditIfNotExists, Disabled	1.0.0
<a href="#">Authentication should be enabled on your Function app</a>	Azure App Service Authentication is a feature that can prevent anonymous HTTP requests from reaching the Function app, or authenticate those that have tokens before they reach the Function app	AuditIfNotExists, Disabled	1.0.0
<a href="#">Authentication should be enabled on your web app</a>	Azure App Service Authentication is a feature that can prevent anonymous HTTP requests from reaching the web app, or authenticate those that have tokens before they reach the web app	AuditIfNotExists, Disabled	1.0.0
<a href="#">Configure App Service slots to disable local authentication for FTP deployments.</a>	Disable local authentication methods for FTP deployments so that your App Services slots exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	DeployIfNotExists, Disabled	1.0.0
<a href="#">Configure App Service slots to disable local authentication for SCM sites.</a>	Disable local authentication methods for SCM sites so that your App Services slots exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	DeployIfNotExists, Disabled	1.0.0
<a href="#">Configure App Service to disable local authentication for SCM sites.</a>	Disable local authentication methods for SCM sites so that your App Services exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	DeployIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Configure App Service to disable local authentication on FTP deployments.</a>	Disable local authentication methods for FTP deployments so that your App Services exclusively require Azure Active Directory identities for authentication. Learn more at: <a href="https://aka.ms/app-service-disable-basic-auth">https://aka.ms/app-service-disable-basic-auth</a> .	DeployIfNotExists, Disabled	1.0.0
<a href="#">Configure App Services to disable public network access</a>	Disable public network access for your App Services so that it is not accessible over the public internet. This can reduce data leakage risks. Learn more at: <a href="https://aka.ms/app-service-private-endpoint">https://aka.ms/app-service-private-endpoint</a> .	DeployIfNotExists, Disabled	1.0.0
<a href="#">CORS should not allow every resource to access your API App</a>	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your API app. Allow only required domains to interact with your API app.	AuditIfNotExists, Disabled	1.0.0
<a href="#">CORS should not allow every resource to access your Function Apps</a>	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your Function app. Allow only required domains to interact with your Function app.	AuditIfNotExists, Disabled	1.0.0
<a href="#">CORS should not allow every resource to access your Web Applications</a>	Cross-Origin Resource Sharing (CORS) should not allow all domains to access your web application. Allow only required domains to interact with your web app.	AuditIfNotExists, Disabled	1.0.0
<a href="#">Diagnostic logs in App Services should be enabled</a>	Audit enabling of diagnostic logs on the app. This enables you to recreate activity trails for investigation purposes if a security incident occurs or your network is compromised	AuditIfNotExists, Disabled	2.0.0
<a href="#">Ensure API app has 'Client Certificates (Incoming client certificates)' set to 'On'</a>	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app.	Audit, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Ensure that 'HTTP Version' is the latest, if used to run the API app</a>	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	2.0.0
<a href="#">Ensure that 'HTTP Version' is the latest, if used to run the Function app</a>	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	2.0.0
<a href="#">Ensure that 'HTTP Version' is the latest, if used to run the Web app</a>	Periodically, newer versions are released for HTTP either due to security flaws or to include additional functionality. Using the latest HTTP version for web apps to take advantage of security fixes, if any, and/or new functionalities of the newer version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	2.0.0
<a href="#">Ensure that 'Java version' is the latest, if used as a part of the API app</a>	Periodically, newer versions are released for Java either due to security flaws or to include additional functionality. Using the latest Python version for API apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	2.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Ensure that 'Java version' is the latest, if used as a part of the Function app</a>	<p>Periodically, newer versions are released for Java software either due to security flaws or to include additional functionality.</p> <p>Using the latest Java version for Function apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.</p>	AuditIfNotExists, Disabled	2.0.0
<a href="#">Ensure that 'Java version' is the latest, if used as a part of the Web app</a>	<p>Periodically, newer versions are released for Java software either due to security flaws or to include additional functionality.</p> <p>Using the latest Java version for web apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.</p>	AuditIfNotExists, Disabled	2.0.0
<a href="#">Ensure that 'PHP version' is the latest, if used as a part of the API app</a>	<p>Periodically, newer versions are released for PHP software either due to security flaws or to include additional functionality.</p> <p>Using the latest PHP version for API apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.</p>	AuditIfNotExists, Disabled	2.1.0
<a href="#">Ensure that 'PHP version' is the latest, if used as a part of the WEB app</a>	<p>Periodically, newer versions are released for PHP software either due to security flaws or to include additional functionality.</p> <p>Using the latest PHP version for web apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.</p>	AuditIfNotExists, Disabled	2.1.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Ensure that 'Python version' is the latest, if used as a part of the API app	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for API apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	3.0.0
Ensure that 'Python version' is the latest, if used as a part of the Function app	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for Function apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	3.0.0
Ensure that 'Python version' is the latest, if used as a part of the Web app	Periodically, newer versions are released for Python software either due to security flaws or to include additional functionality. Using the latest Python version for web apps is recommended in order to take advantage of security fixes, if any, and/or new functionalities of the latest version. Currently, this policy only applies to Linux web apps.	AuditIfNotExists, Disabled	3.0.0
Ensure WEB app has 'Client Certificates (Incoming client certificates)' set to 'On'	Client certificates allow for the app to request a certificate for incoming requests. Only clients that have a valid certificate will be able to reach the app.	Audit, Disabled	1.0.0
FTPS only should be required in your API App	Enable FTPS enforcement for enhanced security	AuditIfNotExists, Disabled	2.0.0
FTPS only should be required in your Function App	Enable FTPS enforcement for enhanced security	AuditIfNotExists, Disabled	2.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
FTPS should be required in your Web App	Enable FTPS enforcement for enhanced security	AuditIfNotExists, Disabled	2.0.0
Function App should only be accessible over HTTPS	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled	1.0.0
Function apps should have 'Client Certificates (Incoming client certificates)' enabled	Client certificates allow for the app to request a certificate for incoming requests. Only clients with valid certificates will be able to reach the app.	Audit, Disabled	1.0.1
Function apps should use an Azure file share for its content directory	The content directory of a function app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to <a href="https://go.microsoft.com/fwlink/?linkid=2151594">https://go.microsoft.com/fwlink/?linkid=2151594</a> .	Audit, Disabled	1.0.0
Latest TLS version should be used in your API App	Upgrade to the latest TLS version	AuditIfNotExists, Disabled	1.0.0
Latest TLS version should be used in your Function App	Upgrade to the latest TLS version	AuditIfNotExists, Disabled	1.0.0
Latest TLS version should be used in your Web App	Upgrade to the latest TLS version	AuditIfNotExists, Disabled	1.0.0
Managed identity should be used in your API App	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	2.0.0
Managed identity should be used in your Function App	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	2.0.0
Managed identity should be used in your Web App	Use a managed identity for enhanced authentication security	AuditIfNotExists, Disabled	2.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Remote debugging should be turned off for API Apps	Remote debugging requires inbound ports to be opened on API apps. Remote debugging should be turned off.	AuditIfExists, Disabled	1.0.0
Remote debugging should be turned off for Function Apps	Remote debugging requires inbound ports to be opened on function apps. Remote debugging should be turned off.	AuditIfExists, Disabled	1.0.0
Remote debugging should be turned off for Web Applications	Remote debugging requires inbound ports to be opened on a web application. Remote debugging should be turned off.	AuditIfExists, Disabled	1.0.0
Resource logs in App Services should be enabled	Audit enabling of resource logs on the app. This enables you to recreate activity trails for investigation purposes if a security incident occurs or your network is compromised.	AuditIfExists, Disabled	1.0.0
Web Application should only be accessible over HTTPS	Use of HTTPS ensures server/service authentication and protects data in transit from network layer eavesdropping attacks.	Audit, Disabled	1.0.0
Web apps should use an Azure file share for its content directory	The content directory of a web app should be located on an Azure file share. The storage account information for the file share must be provided before any publishing activity. To learn more about using Azure Files for hosting app service content refer to <a href="https://go.microsoft.com/fwlink/?linkid=2151594">https://go.microsoft.com/fwlink/?linkid=2151594</a> .	Audit, Disabled	1.0.0

## Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).

# Azure subscription and service limits, quotas, and constraints

11/2/2021 • 122 minutes to read • [Edit Online](#)

This document lists some of the most common Microsoft Azure limits, which are also sometimes called quotas.

To learn more about Azure pricing, see [Azure pricing overview](#). There, you can estimate your costs by using the [pricing calculator](#). You also can go to the pricing details page for a particular service, for example, [Windows VMs](#). For tips to help manage your costs, see [Prevent unexpected costs with Azure billing and cost management](#).

## Managing limits

### NOTE

Some services have adjustable limits.

When a service doesn't have adjustable limits, the following tables use the header **Limit**. In those cases, the default and the maximum limits are the same.

When the limit can be adjusted, the tables include **Default limit** and **Maximum limit** headers. The limit can be raised above the default limit but not above the maximum limit.

If you want to raise the limit or quota above the default limit, [open an online customer support request at no charge](#).

The terms *soft limit* and *hard limit* often are used informally to describe the current, adjustable limit (soft limit) and the maximum limit (hard limit). If a limit isn't adjustable, there won't be a soft limit, only a hard limit.

[Free Trial subscriptions](#) aren't eligible for limit or quota increases. If you have a [Free Trial subscription](#), you can upgrade to a [Pay-As-You-Go](#) subscription. For more information, see [Upgrade your Azure Free Trial subscription to a Pay-As-You-Go subscription](#) and the [Free Trial subscription FAQ](#).

Some limits are managed at a regional level.

Let's use vCPU quotas as an example. To request a quota increase with support for vCPUs, you must decide how many vCPUs you want to use in which regions. You then request an increase in vCPU quotas for the amounts and regions that you want. If you need to use 30 vCPUs in West Europe to run your application there, you specifically request 30 vCPUs in West Europe. Your vCPU quota isn't increased in any other region--only West Europe has the 30-vCPU quota.

As a result, decide what your quotas must be for your workload in any one region. Then request that amount in each region into which you want to deploy. For help in how to determine your current quotas for specific regions, see [Resolve errors for resource quotas](#).

## General limits

For limits on resource names, see [Naming rules and restrictions for Azure resources](#).

For information about Resource Manager API read and write limits, see [Throttling Resource Manager requests](#).

### Management group limits

The following limits apply to [management groups](#).

RESOURCE	LIMIT
Management groups per Azure AD tenant	10,000
Subscriptions per management group	Unlimited.
Levels of management group hierarchy	Root level plus 6 levels <sup>1</sup>
Direct parent management group per management group	One
<a href="#">Management group level deployments</a> per location	800 <sup>2</sup>
Locations of <a href="#">Management group level deployments</a>	10

<sup>1</sup>The 6 levels don't include the subscription level.

<sup>2</sup>If you reach the limit of 800 deployments, delete deployments from the history that are no longer needed. To delete management group level deployments, use [Remove-AzManagementGroupDeployment](#) or [az deployment mg delete](#).

## Subscription limits

The following limits apply when you use Azure Resource Manager and Azure resource groups.

RESOURCE	LIMIT
Subscriptions <a href="#">associated with an Azure Active Directory tenant</a>	Unlimited
<a href="#">Coadministrators</a> per subscription	Unlimited
<a href="#">Resource groups</a> per subscription	980
Azure Resource Manager API request size	4,194,304 bytes
Tags per subscription <sup>1</sup>	50
Unique tag calculations per subscription <sup>1</sup>	80,000
<a href="#">Subscription-level deployments</a> per location	800 <sup>2</sup>
Locations of <a href="#">Subscription-level deployments</a>	10

<sup>1</sup>You can apply up to 50 tags directly to a subscription. However, the subscription can contain an unlimited number of tags that are applied to resource groups and resources within the subscription. The number of tags per resource or resource group is limited to 50. Resource Manager returns a [list of unique tag name and values](#) in the subscription only when the number of tags is 80,000 or less. You still can find a resource by tag when the number exceeds 80,000.

<sup>2</sup>Deployments are automatically deleted from the history as you near the limit. For more information, see [Automatic deletions from deployment history](#).

## Resource group limits

RESOURCE	LIMIT
Resources per <a href="#">resource group</a>	Resources aren't limited by resource group. Instead, they're limited by resource type in a resource group. See next row.
Resources per resource group, per resource type	800 - Some resource types can exceed the 800 limit. See <a href="#">Resources not limited to 800 instances per resource group</a> .
Deployments per resource group in the deployment history	800 <sup>1</sup>
Resources per deployment	800
Management locks per unique <a href="#">scope</a>	20
Number of tags per resource or resource group	50
Tag key length	512
Tag value length	256

<sup>1</sup>Deployments are automatically deleted from the history as you near the limit. Deleting an entry from the deployment history doesn't affect the deployed resources. For more information, see [Automatic deletions from deployment history](#).

#### Template limits

VALUE	LIMIT
Parameters	256
Variables	256
Resources (including copy count)	800
Outputs	64
Template expression	24,576 chars
Resources in exported templates	200
Template size	4 MB
Parameter file size	4 MB

You can exceed some template limits by using a nested template. For more information, see [Use linked templates when you deploy Azure resources](#). To reduce the number of parameters, variables, or outputs, you can combine several values into an object. For more information, see [Objects as parameters](#).

You may get an error with a template or parameter file of less than 4 MB, if the total size of the request is too large. For more information about how to simplify your template to avoid a large request, see [Resolve errors for job size exceeded](#).

## Active Directory limits

Here are the usage constraints and other service limits for the Azure Active Directory (Azure AD) service.

CATEGORY	LIMIT
Tenants	A single user can belong to a maximum of 500 Azure AD tenants as a member or a guest. A single user can create a maximum of 200 directories.
Domains	You can add no more than 5000 managed domain names. If you set up all of your domains for federation with on-premises Active Directory, you can add no more than 2500 domain names in each tenant.
Resources	<ul style="list-style-type: none"><li>A maximum of 50,000 Azure AD resources can be created in a single tenant by users of the Free edition of Azure Active Directory by default. If you have at least one verified domain, the default Azure AD service quota for your organization is extended to 300,000 Azure AD resources. Azure AD service quota for organizations created by self-service sign-up remains 50,000 Azure AD resources even after you performed an internal admin takeover and the organization is converted to a managed tenant with at least one verified domain. This service limit is unrelated to the pricing tier limit of 500,000 resources on the Azure AD pricing page. To go beyond the default quota, you must contact Microsoft Support.</li><li>A non-admin user can create no more than 250 Azure AD resources. Both active resources and deleted resources that are available to restore count toward this quota. Only deleted Azure AD resources that were deleted fewer than 30 days ago are available to restore. Deleted Azure AD resources that are no longer available to restore count toward this quota at a value of one-quarter for 30 days. If you have developers who are likely to repeatedly exceed this quota in the course of their regular duties, you can <a href="#">create and assign a custom role</a> with permission to create a limitless number of app registrations.</li></ul>
Schema extensions	<ul style="list-style-type: none"><li>String-type extensions can have a maximum of 256 characters.</li><li>Binary-type extensions are limited to 256 bytes.</li><li>Only 100 extension values, across <i>all</i> types and <i>all</i> applications, can be written to any single Azure AD resource.</li><li>Only User, Group, TenantDetail, Device, Application, and ServicePrincipal entities can be extended with string-type or binary-type single-valued attributes.</li></ul>

CATEGORY	LIMIT
Applications	<ul style="list-style-type: none"> <li>• A maximum of 100 users can be owners of a single application.</li> <li>• A user, group, or service principal can have a maximum of 1,500 app role assignments. The limitation is on the service principal, user, or group across all app roles and not on a limit on the number of assignments on a single app role.</li> <li>• Password-based single sign-on (SSO) app has a limit of 48 users, which means that there is a limit of 48 keys for username/password pairs per app. If you want to add additional users, see the troubleshooting instructions in <a href="#">Troubleshoot password-based single sign-on in Azure AD</a>.</li> <li>• A user can only have a maximum of 48 apps where they have username and password credentials configured.</li> </ul>
Application Manifest	A maximum of 1200 entries can be added in the Application Manifest.

CATEGORY	LIMIT
Groups	<ul style="list-style-type: none"> <li>A non-admin user can create a maximum of 250 groups in an Azure AD organization. Any Azure AD admin who can manage groups in the organization can also create unlimited number of groups (up to the Azure AD object limit). If you assign a role to remove the limit for a user, assign them to a less privileged built-in role such as User Administrator or Groups Administrator.</li> <li>An Azure AD organization can have a maximum of 5000 dynamic groups.</li> <li>A maximum of 400 role-assignable groups can be created in a single Azure AD organization (tenant).</li> <li>A maximum of 100 users can be owners of a single group.</li> <li>Any number of Azure AD resources can be members of a single group.</li> <li>A user can be a member of any number of groups. Note: when using security groups in combination with SharePoint Online, a user can be a part of 2049 security groups in total (which is transitive, e.g. not only direct group memberships but also indirect group membership). When going over this limit, authentication and search results become unpredictable.</li> <li>By default, the number of members in a group that you can synchronize from your on-premises Active Directory to Azure Active Directory by using Azure AD Connect is limited to 50,000 members. If you need to sync a group membership that's over this limit, you must onboard the <a href="#">Azure AD Connect Sync V2 endpoint API</a>.</li> <li>Nested Groups in Azure AD are not supported within all scenarios</li> <li>Group expiration policy can be assigned to a maximum of 500 Microsoft 365 groups, when selecting a list of groups. There is no limit when the policy is applied to all Microsoft 365 groups.</li> </ul> <p>At this time the following are the supported scenarios with nested groups.</p> <ul style="list-style-type: none"> <li>One group can be added as a member of another group and you can achieve group nesting.</li> <li>Group membership claims (when an app is configured to receive group membership claims in the token, nested groups in which the signed-in user is a member are included)</li> <li>Conditional access (when a conditional access policy has a group scope)</li> <li>Restricting access to self-serve password reset</li> <li>Restricting which users can do Azure AD Join and device registration</li> </ul> <p>The following scenarios DO NOT support nested groups:</p> <ul style="list-style-type: none"> <li>App role assignment (assigning groups to an app is supported, but groups nested within the directly assigned group will not have access), both for access and for provisioning</li> <li>Group-based licensing (assigning a license automatically to all members of a group)</li> </ul>

CATEGORY	LIMIT
Application Proxy	<ul style="list-style-type: none"> <li>Microsoft 365 Groups.</li> </ul> <p>A maximum of 500 transactions per second per App Proxy application</p> <p>A maximum of 750 transactions per second for the Azure AD organization</p> <p>A transaction is defined as a single http request and response for a unique resource. When throttled, clients will receive a 429 response (too many requests).</p>
Access Panel	There's no limit to the number of applications that can be seen in the Access Panel per user regardless of assigned licenses.
Reports	A maximum of 1,000 rows can be viewed or downloaded in any report. Any additional data is truncated.
Administrative units	An Azure AD resource can be a member of no more than 30 administrative units.
Azure AD roles and permissions	<ul style="list-style-type: none"> <li>A maximum of 30 <a href="#">Azure AD custom roles</a> can be created in an Azure AD organization.</li> <li>A maximum of 100 Azure AD custom role assignments for a single principal at tenant scope.</li> <li>A maximum of 100 Azure AD built-in role assignments for a single principal at non-tenant scope (such as administrative unit or Azure AD object). There is no limit for Azure AD built-in role assignments at tenant scope.</li> <li>A group can't be added as a <a href="#">group owner</a>.</li> <li>A user's ability to read other users' tenant information can be restricted only by the Azure AD organization-wide switch to disable all non-admin users' access to all tenant information (not recommended). For more information, see <a href="#">To restrict the default permissions for member users</a>.</li> <li>It may take up to 15 minutes or signing out/signing in before admin role membership additions and revocations take effect.</li> </ul>

## API Management limits

RESOURCE	LIMIT
Maximum number of scale units	12 per region <sup>1</sup>
Cache size	5 GiB per unit <sup>2</sup>
Concurrent back-end connections <sup>3</sup> per HTTP authority	2,048 per unit <sup>4</sup>
Maximum cached response size	2 MiB
Maximum policy document size	256 KiB <sup>5</sup>
Maximum custom gateway domains per service instance <sup>6</sup>	20

RESOURCE	LIMIT
Maximum number of CA certificates per service instance <sup>7</sup>	10
Maximum number of service instances per subscription <sup>8</sup>	20
Maximum number of subscriptions per service instance <sup>8</sup>	500
Maximum number of client certificates per service instance <sup>8</sup>	50
Maximum number of APIs per service instance <sup>8</sup>	50
Maximum number of API management operations per service instance <sup>8</sup>	1,000
Maximum total request duration <sup>8</sup>	30 seconds
Maximum request payload size <sup>8</sup>	1 GiB
Maximum buffered payload size <sup>8</sup>	2 MiB
Maximum request URL size <sup>9</sup>	16384 bytes
Maximum length of URL path segment <sup>10</sup>	260 characters
Maximum size of API schema used by <a href="#">validation policy</a> <sup>10</sup>	4 MB
Maximum size of request or response body in <a href="#">validate-content policy</a> <sup>10</sup>	100 KB
Maximum number of self-hosted gateways <sup>11</sup>	25

<sup>1</sup>Scaling limits depend on the pricing tier. For details on the pricing tiers and their scaling limits, see [API Management pricing](#).

<sup>2</sup>Per unit cache size depends on the pricing tier. To see the pricing tiers and their scaling limits, see [API Management pricing](#).

<sup>3</sup>Connections are pooled and reused unless explicitly closed by the back end.

<sup>4</sup>This limit is per unit of the Basic, Standard, and Premium tiers. The Developer tier is limited to 1,024. This limit doesn't apply to the Consumption tier.

<sup>5</sup>This limit applies to the Basic, Standard, and Premium tiers. In the Consumption tier, policy document size is limited to 16 KiB.

<sup>6</sup>Multiple custom domains are supported in the Developer and Premium tiers only.

<sup>7</sup>CA certificates are not supported in the Consumption tier.

<sup>8</sup>This limit applies to the Consumption tier only. There are no limits in these categories for other tiers.

<sup>9</sup>Applies to the Consumption tier only. Includes an up to 2048 bytes long query string.

<sup>10</sup> To increase this limit, please contact [support](#).

<sup>11</sup>Self-hosted gateways are supported in the Developer and Premium tiers only. The limit applies to the number of [self-hosted gateway resources](#). To raise this limit please contact [support](#). Note, that the number of nodes (or replicas) associated with a self-hosted gateway resource is unlimited in the Premium tier and capped at a single node in the Developer tier.

## App Service limits

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V1-V3)	ISOLATED
Web, mobile, or API apps per Azure App Service plan <sup>1</sup>	10	100	Unlimited <sup>2</sup>	Unlimited <sup>2</sup>	Unlimited <sup>2</sup>	Unlimited <sup>2</sup>
App Service plan	10 per region	10 per resource group	100 per resource group	100 per resource group	100 per resource group	100 per resource group
Compute instance type	Shared	Shared	Dedicated <sup>3</sup>	Dedicated <sup>3</sup>	Dedicated <sup>3</sup>	Dedicated <sup>3</sup>
Scale out (maximum instances)	1 shared	1 shared	3 dedicated <sup>3</sup>	10 dedicated <sup>3</sup>	20 dedicated for v1; 30 dedicated for v2 and v3. <sup>3</sup>	100 dedicated <sup>4</sup>
Storage <sup>5</sup>	1 GB <sup>5</sup>	1 GB <sup>5</sup>	10 GB <sup>5</sup>	50 GB <sup>5</sup>	250 GB <sup>5</sup>	1 TB <sup>5</sup> The available storage quota is 999 GB.
CPU time (5 minutes) <sup>6</sup>	3 minutes	3 minutes	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates
CPU time (day) <sup>6</sup>	60 minutes	240 minutes	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates	Unlimited, pay at standard rates
Memory (1 hour)	1,024 MB per App Service plan	1,024 MB per app	N/A	N/A	N/A	N/A
Bandwidth	165 MB	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply	Unlimited, data transfer rates apply
Application architecture	32-bit	32-bit	32-bit/64-bit	32-bit/64-bit	32-bit/64-bit	32-bit/64-bit
Web sockets per instance <sup>7</sup>	5	35	350	Unlimited	Unlimited	Unlimited
Outbound IP connections per instance	600	600	Depends on instance size <sup>8</sup>	Depends on instance size <sup>8</sup>	Depends on instance size <sup>8</sup>	16,000

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V1-V3)	ISOLATED
Concurrent debugger connections per application	1	1	1	5	5	5
App Service Certificates per subscription <sup>9</sup>	Not supported	Not supported	10	10	10	10
Custom domains per app	0 (azurewebsites.net subdomain only)	500	500	500	500	500
Custom domain SSL support	Not supported, wildcard certificate for *.azurewebsites.net available by default	Not supported, wildcard certificate for *.azurewebsites.net available by default	Unlimited SNI SSL connections	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included	Unlimited SNI SSL and 1 IP SSL connections included
Hybrid connections			5 per plan	25 per plan	220 per app	220 per app
Virtual Network Integration				X	X	X
Private Endpoints					100 per app	
Integrated load balancer		X	X	X	X	X <sup>10</sup>
Access restrictions	512 rules per app	512 rules per app	512 rules per app	512 rules per app	512 rules per app	512 rules per app
Always On			X	X	X	X
Scheduled backups				Scheduled backups every 2 hours, a maximum of 12 backups per day (manual + scheduled)	Scheduled backups every hour, a maximum of 50 backups per day (manual + scheduled)	Scheduled backups every hour, a maximum of 50 backups per day (manual + scheduled)
Autoscale				X	X	X

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V1-V3)	ISOLATED
WebJobs <sup>11</sup>	X	X	X	X	X	X
Endpoint monitoring			X	X	X	X
Staging slots per app				5	20	20
Testing in Production				X	X	X
Diagnostic Logs	X	X	X	X	X	X
Kudu	X	X	X	X	X	X
Authentication and Authorization	X	X	X	X	X	X
App Service Managed Certificates (Public Preview) <sup>12</sup>			X	X	X	X
SLA			99.95%	99.95%	99.95%	99.95%

<sup>1</sup> Apps and storage quotas are per App Service plan unless noted otherwise.

<sup>2</sup> The actual number of apps that you can host on these machines depends on the activity of the apps, the size of the machine instances, and the corresponding resource utilization.

<sup>3</sup> Dedicated instances can be of different sizes. For more information, see [App Service pricing](#).

<sup>4</sup> More are allowed upon request.

<sup>5</sup> The storage limit is the total content size across all apps in the same App service plan. The total content size of all apps across all App service plans in a single resource group and region cannot exceed 500 GB. The file system quota for App Service hosted apps is determined by the aggregate of App Service plans created in a region and resource group.

<sup>6</sup> These resources are constrained by physical resources on the dedicated instances (the instance size and the number of instances).

<sup>7</sup> If you scale an app in the Basic tier to two instances, you have 350 concurrent connections for each of the two instances. For Standard tier and above, there are no theoretical limits to web sockets, but other factors can limit the number of web sockets. For example, maximum concurrent requests allowed (defined by `maxConcurrentRequestsPerCpu`) are: 7,500 per small VM, 15,000 per medium VM (7,500 x 2 cores), and 75,000 per large VM (18,750 x 4 cores).

<sup>8</sup> The maximum IP connections are per instance and depend on the instance size: 1,920 per B1/S1/P1V3 instance, 3,968 per B2/S2/P2V3 instance, 8,064 per B3/S3/P3V3 instance.

<sup>9</sup> The App Service Certificate quota limit per subscription can be increased via a support request to a maximum

limit of 200.

<sup>10</sup> App Service Isolated SKUs can be internally load balanced (ILB) with Azure Load Balancer, so there's no public connectivity from the internet. As a result, some features of an ILB Isolated App Service must be used from machines that have direct access to the ILB network endpoint.

<sup>11</sup> Run custom executables and/or scripts on demand, on a schedule, or continuously as a background task within your App Service instance. Always On is required for continuous WebJobs execution. There's no predefined limit on the number of WebJobs that can run in an App Service instance. There are practical limits that depend on what the application code is trying to do.

<sup>12</sup> Naked domains aren't supported. Only issuing standard certificates (wildcard certificates aren't available). Limited to only one free certificate per custom domain.

## Automation limits

### Process automation

RESOURCE	LIMIT	NOTES
Maximum number of new jobs that can be submitted every 30 seconds per Azure Automation account (nonscheduled jobs)	100	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response.
Maximum number of concurrent running jobs at the same instance of time per Automation account (nonscheduled jobs)	200	When this limit is reached, the subsequent requests to create a job fail. The client receives an error response.
Maximum storage size of job metadata for a 30-day rolling period	10 GB (approximately 4 million jobs)	When this limit is reached, the subsequent requests to create a job fail.
Maximum job stream limit	1 MiB	A single stream cannot be larger than 1 MiB.
Maximum number of modules that can be imported every 30 seconds per Automation account	5	
Maximum size of a module	100 MB	
Maximum size of a node configuration file	1 MB	Applies to state configuration
Job run time, Free tier	500 minutes per subscription per calendar month	
Maximum amount of disk space allowed per sandbox <sup>1</sup>	1 GB	Applies to Azure sandboxes only.
Maximum amount of memory given to a sandbox <sup>1</sup>	400 MB	Applies to Azure sandboxes only.
Maximum number of network sockets allowed per sandbox <sup>1</sup>	1,000	Applies to Azure sandboxes only.

RESOURCE	LIMIT	NOTES
Maximum runtime allowed per runbook <sup>1</sup>	3 hours	Applies to Azure sandboxes only.
Maximum number of Automation accounts in a subscription	No limit	
Maximum number of system hybrid runbook workers per Automation Account	4,000	
Maximum number of user hybrid runbook workers per Automation Account	4,000	
Maximum number of concurrent jobs that can be run on a single Hybrid Runbook Worker	50	
Maximum runbook job parameter size	512 kilobytes	
Maximum runbook parameters	50	If you reach the 50-parameter limit, you can pass a JSON or XML string to a parameter and parse it with the runbook.
Maximum webhook payload size	512 kilobytes	
Maximum days that job data is retained	30 days	
Maximum PowerShell workflow state size	5 MB	Applies to PowerShell workflow runbooks when checkpointing workflow.
Maximum number of tags supported by an Automation account	15	

<sup>1</sup>A sandbox is a shared environment that can be used by multiple jobs. Jobs that use the same sandbox are bound by the resource limitations of the sandbox.

#### Change Tracking and Inventory

The following table shows the tracked item limits per machine for change tracking.

RESOURCE	LIMIT	NOTES
File	500	
File size	5 MB	
Registry	250	
Windows software	250	Doesn't include software updates.

RESOURCE	LIMIT	NOTES
Linux packages	1,250	
Services	250	
Daemon	250	

#### Update Management

The following table shows the limits for Update Management.

RESOURCE	LIMIT	NOTES
Number of machines per update deployment	1000	
Number of dynamic groups per update deployment	500	

## Azure App Configuration

RESOURCE	LIMIT	COMMENT
Configuration stores for Free tier	1 store per subscription	
Configuration stores for Standard tier	Unlimited stores per subscription	
Configuration store requests for Free tier	1,000 requests per day	Once the quota is exhausted, HTTP status code 429 will be returned for all requests until the end of the day
Configuration store requests for Standard tier	30,000 per hour	Once the quota is exhausted, requests may return HTTP status code 429 indicating Too Many Requests - until the end of the hour
Storage for Free tier	10 MB	
Storage for Standard tier	1 GB	
Keys and Values	10 KB	For a single key-value item, including all metadata

## Azure API for FHIR service limits

Azure API for FHIR is a managed, standards-based, compliant API for clinical health data that enables solutions for actionable analytics and machine learning.

QUOTA NAME	DEFAULT LIMIT	MAXIMUM LIMIT	NOTES
Request Units (RUs)	10,000 RUs	Contact support Maximum available is 1,000,000.	You need a minimum of 400 RUs or 40 RUs/GB, whichever is larger.

Quota name	Default limit	Maximum limit	Notes
Concurrent connections	15 concurrent connections on two instances (for a total of 30 concurrent requests)	Contact support	
Azure API for FHIR Service Instances per Subscription	10	Contact support	

## Azure Cache for Redis limits

Resource	Limit
Cache size	1.2 TB
Databases	64
Maximum connected clients	40,000
Azure Cache for Redis replicas, for high availability	1
Shards in a premium cache with clustering	10

Azure Cache for Redis limits and sizes are different for each pricing tier. To see the pricing tiers and their associated sizes, see [Azure Cache for Redis pricing](#).

For more information on Azure Cache for Redis configuration limits, see [Default Redis server configuration](#).

Because configuration and management of Azure Cache for Redis instances is done by Microsoft, not all Redis commands are supported in Azure Cache for Redis. For more information, see [Redis commands not supported in Azure Cache for Redis](#).

## Azure Cloud Services limits

Resource	Limit
<a href="#">Web or worker roles per deployment</a> <sup>1</sup>	25
<a href="#">Instance input endpoints</a> per deployment	25
<a href="#">Input endpoints</a> per deployment	25
<a href="#">Internal endpoints</a> per deployment	25
<a href="#">Hosted service certificates</a> per deployment	199

<sup>1</sup>Each Azure Cloud Service with web or worker roles can have two deployments, one for production and one for staging. This limit refers to the number of distinct roles, that is, configuration. This limit doesn't refer to the number of instances per role, that is, scaling.

## Azure Cognitive Search limits

Pricing tiers determine the capacity and limits of your search service. Tiers include:

- **Free** multi-tenant service, shared with other Azure subscribers, is intended for evaluation and small development projects.
- **Basic** provides dedicated computing resources for production workloads at a smaller scale, with up to three replicas for highly available query workloads.
- **Standard**, which includes S1, S2, S3, and S3 High Density, is for larger production workloads. Multiple levels exist within the Standard tier so that you can choose a resource configuration that best matches your workload profile.

## Limits per subscription

You can create multiple services within a subscription. Each one can be provisioned at a specific tier. You're limited only by the number of services allowed at each tier. For example, you could create up to 12 services at the Basic tier and another 12 services at the S1 tier within the same subscription. For more information about tiers, see [Choose an SKU or tier for Azure Cognitive Search](#).

Maximum service limits can be raised upon request. If you need more services within the same subscription, contact Azure Support.

RESOURCE	FREE <sup>1</sup>	BASIC	S1	S2	S3	S3 HD	L1	L2
Maximum services	1	16	16	8	6	6	6	6
Maximum scale in search units (SU) <sup>2</sup>	N/A	3 SU	36 SU	36 SU	36 SU	36 SU	36 SU	36 SU

<sup>1</sup> Free is based on shared, not dedicated, resources. Scale-up is not supported on shared resources.

<sup>2</sup> Search units are billing units, allocated as either a *replica* or a *partition*. You need both resources for storage, indexing, and query operations. To learn more about SU computations, see [Scale resource levels for query and index workloads](#).

## Limits per search service

A search service is constrained by disk space or by a hard limit on the maximum number of indexes or indexers, whichever comes first. The following table documents storage limits. For maximum object limits, see [Limits by resource](#).

RESOURCE	FREE	BASIC <sup>1</sup>	S1	S2	S3	S3 HD	L1	L2
Service level agreement (SLA) <sup>2</sup>	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Storage per partition	50 MB	2 GB	25 GB	100 GB	200 GB	200 GB	1 TB	2 TB
Partitions per service	N/A	1	12	12	12	3	12	12

RESOURCE	FREE	BASIC	S1	S2	S3	S3 HD	L1	L2
Partition size	N/A	2 GB	25 GB	100 GB	200 GB	200 GB	1 TB	2 TB
Replicas	N/A	3	12	12	12	12	12	12

<sup>1</sup> Basic has one fixed partition. Additional search units can be used to add replicas for larger query volumes.

<sup>2</sup> Service level agreements are in effect for billable services on dedicated resources. Free services and preview features have no SLA. For billable services, SLAs take effect when you provision sufficient redundancy for your service. Two or more replicas are required for query (read) SLAs. Three or more replicas are required for query and indexing (read-write) SLAs. The number of partitions isn't an SLA consideration.

To learn more about limits on a more granular level, such as document size, queries per second, keys, requests, and responses, see [Service limits in Azure Cognitive Search](#).

## Azure Cognitive Services limits

The following limits are for the number of Cognitive Services resources per Azure subscription. Each of the Cognitive Services may have other limitations, for more information, see [Azure Cognitive Services](#).

TYPE	LIMIT	EXAMPLE
A mixture of Cognitive Services resources	Maximum of 200 total Cognitive Services resources per region.	100 Computer Vision resources in West US, 50 Speech Service resources in West US, and 50 Text Analytics resources in West US.
A single type of Cognitive Services resources.	Maximum of 100 resources per region	100 Computer Vision resources in West US 2, and 100 Computer Vision resources in East US.

## Azure Cosmos DB limits

For Azure Cosmos DB limits, see [Limits in Azure Cosmos DB](#).

## Azure Data Explorer limits

The following table describes the maximum limits for Azure Data Explorer clusters.

RESOURCE	LIMIT
Clusters per region per subscription	20
Instances per cluster	1000
Number of databases in a cluster	10,000
Number of follower clusters (data share consumers) per leader cluster (data share producer)	100

The following table describes the limits on management operations performed on Azure Data Explorer clusters.

SCOPE	OPERATION	LIMIT
Cluster	read (for example, get a cluster)	500 per 5 minutes
Cluster	write (for example, create a database)	1000 per hour

## Azure Database for MySQL

For Azure Database for MySQL limits, see [Limitations in Azure Database for MySQL](#).

## Azure Database for PostgreSQL

For Azure Database for PostgreSQL limits, see [Limitations in Azure Database for PostgreSQL](#).

## Azure Functions limits

RESOURCE	CONSUMPTION PLAN	PREMIUM PLAN	DEDICATED PLAN	ASE	KUBERNETES
Default <a href="#">timeout duration</a> (min)	5	30	30 <sup>1</sup>	30	30
Max <a href="#">timeout duration</a> (min)	10	unbounded <sup>7</sup>	unbounded <sup>2</sup>	unbounded	unbounded
Max outbound connections (per instance)	600 active (1200 total)	unbounded	unbounded	unbounded	unbounded
Max request size (MB) <sup>3</sup>	100	100	100	100	Depends on cluster
Max query string length <sup>3</sup>	4096	4096	4096	4096	Depends on cluster
Max request URL length <sup>3</sup>	8192	8192	8192	8192	Depends on cluster
ACU per instance	100	210-840	100-840	210-250 <sup>8</sup>	<a href="#">AKS pricing</a>
Max memory (GB per instance)	1.5	3.5-14	1.75-14	3.5 - 14	Any node is supported
Max instance count	200	100 <sup>9</sup>	varies by SKU <sup>10</sup>	100 <sup>10</sup>	Depends on cluster
Function apps per plan	100	100	unbounded <sup>4</sup>	unbounded	unbounded
<a href="#">App Service plans</a>	100 per <a href="#">region</a>	100 per resource group	100 per resource group	-	-
Storage <sup>5</sup>	5 TB	250 GB	50-1000 GB	1 TB	n/a

RESOURCE	CONSUMPTION PLAN	PREMIUM PLAN	DEDICATED PLAN	ASE	KUBERNETES
Custom domains per app	500 <sup>6</sup>	500	500	500	n/a
Custom domain SSL support	unbounded SNI SSL connection included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included	n/a

<sup>1</sup> By default, the timeout for the Functions 1.x runtime in an App Service plan is unbounded.

<sup>2</sup> Requires the App Service plan be set to [Always On](#). Pay at standard [rates](#).

<sup>3</sup> These limits are [set in the host](#).

<sup>4</sup> The actual number of function apps that you can host depends on the activity of the apps, the size of the machine instances, and the corresponding resource utilization.

<sup>5</sup> The storage limit is the total content size in temporary storage across all apps in the same App Service plan. Consumption plan uses Azure Files for temporary storage.

<sup>6</sup> When your function app is hosted in a [Consumption plan](#), only the CNAME option is supported. For function apps in a [Premium plan](#) or an [App Service plan](#), you can map a custom domain using either a CNAME or an A record.

<sup>7</sup> Guaranteed for up to 60 minutes.

<sup>8</sup> Workers are roles that host customer apps. Workers are available in three fixed sizes: One vCPU/3.5 GB RAM; Two vCPU/7 GB RAM; Four vCPU/14 GB RAM.

<sup>9</sup> When running on Linux in a Premium plan, you're currently limited to 20 instances.

<sup>10</sup> See [App Service limits](#) for details.

For more information, see [Functions Hosting plans comparison](#).

## Azure Kubernetes Service limits

RESOURCE	LIMIT
Maximum clusters per subscription	5000
Maximum nodes per cluster with Virtual Machine Availability Sets and Basic Load Balancer SKU	100
Maximum nodes per cluster with Virtual Machine Scale Sets and <a href="#">Standard Load Balancer SKU</a>	1000 (across all <a href="#">node pools</a> )
Maximum node pools per cluster	100
Maximum pods per node: <a href="#">Basic networking</a> with Kubelet	Maximum: 250 Azure CLI default: 110 Azure Resource Manager template default: 110 Azure portal deployment default: 30
Maximum pods per node: <a href="#">Advanced networking</a> with Azure Container Networking Interface	Maximum: 250 Default: 30
Open Service Mesh (OSM) AKS addon preview	Kubernetes Cluster Version: 1.19+ <sup>1</sup> OSM controllers per cluster: 1 <sup>1</sup> Pods per OSM controller: 500 <sup>1</sup> Kubernetes service accounts managed by OSM: 50 <sup>1</sup>

<sup>1</sup> The OSM add-on for AKS is in a preview state and will undergo additional enhancements before general availability (GA). During the preview phase, it's recommended to not surpass the limits shown.

KUBERNETES CONTROL PLANE TIER	LIMIT
Paid tier	Automatically scales out based on the load
Free tier	Limited resources with <a href="#">inflight requests limit</a> of 50 mutating and 100 read-only calls

## Azure Machine Learning limits

The latest values for Azure Machine Learning Compute quotas can be found in the [Azure Machine Learning quota page](#)

## Azure Maps limits

The following table shows the usage limit for the Azure Maps S0 pricing tier. Usage limit depends on the pricing tier.

RESOURCE	S0 PRICING TIER LIMIT
Maximum request rate per subscription	50 requests per second

The following table shows the cumulative data size limit for Azure Maps accounts in an Azure subscription. The Azure Maps Data service is available only at the S1 pricing tier.

RESOURCE	LIMIT
Maximum storage per Azure subscription	1 GB
Maximum size per file upload	100 MB

For more information on the Azure Maps pricing tiers, see [Azure Maps pricing](#).

## Azure Monitor limits

### Alerts

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Metric alerts (classic)	100 active alert rules per subscription.	Call support
Metric alerts	5,000 active alert rules per subscription in Azure public, Azure China 21Vianet and Azure Government clouds. If you are hitting this limit, explore if you can use <a href="#">same type multi-resource alerts</a> . 5,000 metric time-series per alert rule.	Call support.
Activity log alerts	100 active alert rules per subscription (cannot be increased).	Same as default

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Log alerts	1000 active alert rules per subscription. 1000 active alert rules per resource.	Call support
Alert rules and Action rules description length	Log search alerts 4096 characters All other 2048 characters	Same as default

## Alerts API

Azure Monitor Alerts have several throttling limits to protect against users making an excessive number of calls. Such behavior can potentially overload the system backend resources and jeopardize service responsiveness. The following limits are designed to protect customers from interruptions and ensure consistent service level. The user throttling and limits are designed to impact only extreme usage scenario and should not be relevant for typical usage.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
GET alertsSummary	50 calls per minute per subscription	Same as default
GET alerts (without specifying an alert ID)	100 calls per minute per subscription	Same as default
All other calls	1000 calls per minute per subscription	Same as default

## Action groups

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure app push	10 Azure app actions per action group.	Same as Default
Email	1,000 email actions in an action group. No more than 100 emails in an hour. Also see the <a href="#">rate limiting information</a> .	Same as Default
ITSM	10 ITSM actions in an action group.	Same as Default
Logic app	10 logic app actions in an action group.	Same as Default
Runbook	10 runbook actions in an action group.	Same as Default
SMS	10 SMS actions in an action group. No more than 1 SMS message every 5 minutes. Also see the <a href="#">rate limiting information</a> .	Same as Default
Voice	10 voice actions in an action group. No more than 1 voice call every 5 minutes. Also see the <a href="#">rate limiting information</a> .	Same as Default

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Webhook	10 webhook actions in an action group. Maximum number of webhook calls is 1500 per minute per subscription. Other limits are available at <a href="#">action-specific information</a> .	Same as Default

## Autoscale

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Autoscale settings	100 per region per subscription.	Same as default
Autoscale profiles	20 profiles per autoscale setting.	Same as default

## Log queries and language

### General query limits

LIMIT	DESCRIPTION
Query language	Azure Monitor uses the same <a href="#">Kusto query language</a> as Azure Data Explorer. See <a href="#">Azure Monitor log query language differences</a> for KQL language elements not supported in Azure Monitor.
Azure regions	Log queries can experience excessive overhead when data spans Log Analytics workspaces in multiple Azure regions. See <a href="#">Query limits</a> for details.
Cross resource queries	Maximum number of Application Insights resources and Log Analytics workspaces in a single query limited to 100. Cross-resource query is not supported in View Designer. Cross-resource query in log alerts is supported in the new scheduledQueryRules API. See <a href="#">Cross-resource query limits</a> for details.

### User query throttling

Azure Monitor has several throttling limits to protect against users sending an excessive number of queries. Such behavior can potentially overload the system backend resources and jeopardize service responsiveness. The following limits are designed to protect customers from interruptions and ensure consistent service level. The user throttling and limits are designed to impact only extreme usage scenario and should not be relevant for typical usage.

MEASURE	LIMIT PER USER	DESCRIPTION
Concurrent queries	5	If there are already 5 queries running for the user, any new queries are placed in a per-user concurrency queue. When one of the running queries ends, the next query will be pulled from the queue and started. This does not include queries from alert rules.

Measure	Limit per user	Description
Time in concurrency queue	3 minutes	If a query sits in the queue for more than 3 minutes without being started, it will be terminated with an HTTP error response with code 429.
Total queries in concurrency queue	200	Once the number of queries in the queue reaches 200, any additional queries will be rejected with an HTTP error code 429. This number is in addition to the 5 queries that can be running simultaneously.
Query rate	200 queries per 30 seconds	This is the overall rate that queries can be submitted by a single user to all workspaces. This limit applies to programmatic queries or queries initiated by visualization parts such as Azure dashboards and the Log Analytics workspace summary page.

- Optimize your queries as described in [Optimize log queries in Azure Monitor](#).
- Dashboards and workbooks can contain multiple queries in a single view that generate a burst of queries every time they load or refresh. Consider breaking them up into multiple views that load on demand.
- In Power BI, consider extracting only aggregated results rather than raw logs.

## Log Analytics workspaces

### Data collection volume and retention

Tier	Limit per day	Data retention	Comment
Current Per GB pricing tier (introduced April 2018)	No limit	30 - 730 days	Data retention beyond 31 days is available for additional charges. Learn more about Azure Monitor pricing.
Legacy Free tiers (introduced April 2016)	500 MB	7 days	When your workspace reaches the 500 MB per day limit, data ingestion stops and resumes at the start of the next day. A day is based on UTC. Note that data collected by Azure Security Center is not included in this 500 MB per day limit and will continue to be collected above this limit.
Legacy Standalone Per GB tier (introduced April 2016)	No limit	30 to 730 days	Data retention beyond 31 days is available for additional charges. Learn more about Azure Monitor pricing.

TIER	LIMIT PER DAY	DATA RETENTION	COMMENT
Legacy Per Node (OMS) (introduced April 2016)	No limit	30 to 730 days	Data retention beyond 31 days is available for additional charges. Learn more about Azure Monitor pricing.
Legacy Standard tier	No limit	30 days	Retention can't be adjusted
Legacy Premium tier	No limit	365 days	Retention can't be adjusted

#### Number of workspaces per subscription.

PRICING TIER	WORKSPACE LIMIT	COMMENTS
Free tier	10	This limit can't be increased.
All other tiers	No limit	You're limited by the number of resources within a resource group and the number of resource groups per subscription.

#### Azure portal

CATEGORY	LIMIT	COMMENTS
Maximum records returned by a log query	30,000	Reduce results using query scope, time range, and filters in the query.

#### Data Collector API

CATEGORY	LIMIT	COMMENTS
Maximum size for a single post	30 MB	Split larger volumes into multiple posts.
Maximum size for field values	32 KB	Fields longer than 32 KB are truncated.

#### Query API

CATEGORY	LIMIT	COMMENTS
Maximum records returned in a single query	500,000	
Maximum size of data returned	~104 MB (~100 MiB)	
Maximum query running time	10 minutes	See <a href="#">Timeouts</a> for details.
Maximum request rate	200 requests per 30 seconds per Azure AD user or client IP address	See <a href="#">Rate limits</a> for details.

#### Azure Monitor Logs connector

CATEGORY	LIMIT	COMMENTS
Max size of data	~16.7 MB (~16 MiB)	Connector infrastructure dictates that limit is set lower than query API limit
Max number of records	500,000	
Max query timeout	110 second	
Charts		Visualization in Logs page and the connector are using different charting libraries and some functionality isn't available in the connector currently.

## General workspace limits

CATEGORY	LIMIT	COMMENTS
Maximum columns in a table	500	
Maximum characters for column name	500	

## Data ingestion volume rate

Azure Monitor is a high scale data service that serves thousands of customers sending terabytes of data each month at a growing pace. The volume rate limit intends to isolate Azure Monitor customers from sudden ingestion spikes in multitenancy environment. A default ingestion volume rate threshold of 500 MB (compressed) is defined in workspaces, this is translated to approximately **6 GB/min** uncompressed -- the actual size can vary between data types depending on the log length and its compression ratio. The volume rate limit applies to data ingested from Azure resources via [Diagnostic settings](#). When volume rate limit is reached, a retry mechanism attempts to ingest the data 4 times in a period of 30 minutes and drop it if operation fails. It doesn't apply to data ingested from [agents](#) or [Data Collector API](#).

When data sent to your workspace is at a volume rate higher than 80% of the threshold configured in your workspace, an event is sent to the *Operation* table in your workspace every 6 hours while the threshold continues to be exceeded. When ingested volume rate is higher than threshold, some data is dropped and an event is sent to the *Operation* table in your workspace every 6 hours while the threshold continues to be exceeded. If your ingestion volume rate continues to exceed the threshold or you are expecting to reach it sometime soon, you can request to increase it in by opening a support request.

See [Monitor health of Log Analytics workspace in Azure Monitor](#) to create alert rules to be proactively notified when you reach any ingestion limits.

### NOTE

Depending on how long you've been using Log Analytics, you might have access to legacy pricing tiers. Learn more about [Log Analytics legacy pricing tiers](#).

## Application Insights

There are some limits on the number of metrics and events per application, that is, per instrumentation key. Limits depend on the [pricing plan](#) that you choose.

RESOURCE	DEFAULT LIMIT	NOTE
Total data per day	100 GB	You can reduce data by setting a cap. If you need more data, you can increase the limit in the portal, up to 1,000 GB. For capacities greater than 1,000 GB, send email to <a href="mailto:AIDataCap@microsoft.com">AIDataCap@microsoft.com</a> .
Throttling	32,000 events/second	The limit is measured over a minute.
Data retention Logs	<a href="#">30 - 730 days</a>	This resource is for <a href="#">Logs</a> .
Data retention Metrics	90 days	This resource is for <a href="#">Metrics Explorer</a> .
<a href="#">Availability multi-step test</a> detailed results retention	90 days	This resource provides detailed results of each step.
Maximum telemetry item size	64 kB	
Maximum telemetry items per batch	64 K	
Property and metric name length	150	See <a href="#">type schemas</a> .
Property value string length	8,192	See <a href="#">type schemas</a> .
Trace and exception message length	32,768	See <a href="#">type schemas</a> .
<a href="#">Availability tests</a> count per app	100	
<a href="#">Profiler</a> data retention	5 days	
<a href="#">Profiler</a> data sent per day	10 GB	

For more information, see [About pricing and quotas in Application Insights](#).

## Azure Data Factory limits

Azure Data Factory is a multitenant service that has the following default limits in place to make sure customer subscriptions are protected from each other's workloads. To raise the limits up to the maximum for your subscription, contact support.

### Version 2

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a data factory	5,000	<a href="#">Contact support</a> .
Total CPU cores for Azure-SSIS Integration Runtimes under one subscription	256	<a href="#">Contact support</a> .

Resource	Default Limit	Maximum Limit
Concurrent pipeline runs per data factory that's shared among all pipelines in the factory	10,000	10,000
Concurrent External activity runs per subscription per <a href="#">Azure Integration Runtime region</a>  External activities are managed on integration runtime but execute on linked services, including Databricks, stored procedure, Web, and others. This limit does not apply to Self-hosted IR.	3,000	3,000
Concurrent Pipeline activity runs per subscription per <a href="#">Azure Integration Runtime region</a>  Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete. This limit does not apply to Self-hosted IR.	1,000	1,000
Concurrent authoring operations per subscription per <a href="#">Azure Integration Runtime region</a>  Including test connection, browse folder list and table list, preview data. This limit does not apply to Self-hosted IR.	200	200
Concurrent Data Integration Units <sup>1</sup> consumption per subscription per <a href="#">Azure Integration Runtime region</a>	Region group 1 <sup>2</sup> : 6,000 Region group 2 <sup>2</sup> : 3,000 Region group 3 <sup>2</sup> : 1,500 Managed virtual network <sup>2</sup> : 2,400	Region group 1 <sup>2</sup> : 6,000 Region group 2 <sup>2</sup> : 3,000 Region group 3 <sup>2</sup> : 1,500 Managed virtual network: <a href="#">Contact support</a> .
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	<a href="#">Contact support</a> .
Maximum parameters per pipeline	50	50
ForEach items	100,000	100,000
ForEach parallelism	20	50
Maximum queued runs per pipeline	100	100
Characters per expression	8,192	8,192
Minimum tumbling window trigger interval	5 min	15 min

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum timeout for pipeline activity runs	7 days	7 days
Bytes per object for pipeline objects <sup>3</sup>	200 KB	200 KB
Bytes per object for dataset and linked service objects <sup>3</sup>	100 KB	2,000 KB
Bytes per payload for each activity run <sup>4</sup>	896 KB	896 KB
Data Integration Units <sup>1</sup> per copy activity run	256	256
Write API calls	1,200/h	1,200/h  This limit is imposed by Azure Resource Manager, not Azure Data Factory.
Read API calls	12,500/h	12,500/h  This limit is imposed by Azure Resource Manager, not Azure Data Factory.
Monitoring queries per minute	1,000	1,000
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	<a href="#">Contact support.</a>
Concurrent number of data flows per integration runtime in managed vNet	20	<a href="#">Contact support.</a>
Concurrent number of data flow debug sessions per user per factory	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a factory	2 GB	<a href="#">Contact support.</a>

<sup>1</sup> The data integration unit (DIU) is used in a cloud-to-cloud copy operation, learn more from [Data integration units \(version 2\)](#). For information on billing, see [Azure Data Factory pricing](#).

<sup>2</sup> [Azure Integration Runtime](#) is [globally available](#) to ensure data compliance, efficiency, and reduced network egress costs.

REGION GROUP	REGIONS
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2

Region group	Regions
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

If managed virtual network is enabled, the data integration unit (DIU) in all region groups are 2,400.

<sup>3</sup> Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

<sup>4</sup> The payload for each activity run includes the activity configuration, the associated dataset(s) and linked service(s) configurations if any, and a small portion of system properties generated per activity type. Limit for this payload size doesn't relate to the amount of data you can move and process with Azure Data Factory. Learn about the [symptoms and recommendation](#) if you hit this limit.

## Version 1

Resource	Default limit	Maximum limit
Pipelines within a data factory	2,500	<a href="#">Contact support</a> .
Data sets within a data factory	5,000	<a href="#">Contact support</a> .
Concurrent slices per data set	10	10
Bytes per object for pipeline objects <sup>1</sup>	200 KB	200 KB
Bytes per object for data set and linked service objects <sup>1</sup>	100 KB	2,000 KB
Azure HDInsight on-demand cluster cores within a subscription <sup>2</sup>	60	<a href="#">Contact support</a> .
Cloud data movement units per copy activity run <sup>3</sup>	32	32
Retry count for pipeline activity runs	1,000	MaxInt (32 bit)

<sup>1</sup> Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

<sup>2</sup> On-demand HDInsight cores are allocated out of the subscription that contains the data factory. As a result, the previous limit is the Data Factory-enforced core limit for on-demand HDInsight cores. It's different from the core limit that's associated with your Azure subscription.

<sup>3</sup> The cloud data movement unit (DMU) for version 1 is used in a cloud-to-cloud copy operation, learn more from [Cloud data movement units \(version 1\)](#). For information on billing, see [Azure Data Factory pricing](#).

RESOURCE	DEFAULT LOWER LIMIT	MINIMUM LIMIT
Scheduling interval	15 minutes	15 minutes
Interval between retry attempts	1 second	1 second
Retry timeout value	1 second	1 second

#### Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource Manager API limits](#).

## Azure NetApp Files

Azure NetApp Files has a regional limit for capacity. The standard capacity limit for each subscription is 25 TiB, per region, across all service levels. To increase the capacity, use the [Service and subscription limits \(quotas\)](#) support request.

To learn more about the limits for Azure NetApp Files, see [Resource limits for Azure NetApp Files](#).

## Azure Policy limits

There's a maximum count for each object type for Azure Policy. For definitions, an entry of *Scope* means the [management group](#) or subscription. For assignments and exemptions, an entry of *Scope* means the [management group](#), subscription, resource group, or individual resource.

WHERE	WHAT	MAXIMUM COUNT
Scope	Policy definitions	500
Scope	Initiative definitions	200
Tenant	Initiative definitions	2,500
Scope	Policy or initiative assignments	200
Scope	Exemptions	1000
Policy definition	Parameters	20
Initiative definition	Policies	1000
Initiative definition	Parameters	300
Policy or initiative assignments	Exclusions (notScopes)	400
Policy rule	Nested conditionals	512
Remediation task	Resources	500

## Azure Quantum limits

### Provider Limits & Quota

The Azure Quantum Service supports both first and third-party service providers. Third-party providers own their limits and quotas. Users can view offers and limits in the Azure portal when configuring third-party providers.

You can find the published quota limits for Microsoft's first party Optimization Solutions provider below.

#### Learn & Develop SKU

RESOURCE	LIMIT
CPU-based concurrent jobs	up to $5^1$ concurrent jobs
FPGA-based concurrent jobs	up to $2^1$ concurrent jobs
CPU-based solver hours	20 hours per month
FPGA-based solver hours	1 hour per month

While on the Learn & Develop SKU, you **cannot** request an increase on your quota limits. Instead you should switch to the Performance at Scale SKU.

#### Performance at Scale SKU

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
CPU-based concurrent jobs	up to $100^1$ concurrent jobs	same as default limit
FPGA-based concurrent jobs	up to $10^1$ concurrent jobs	same as default limit
Solver hours	1,000 hours per month	up to 50,000 hours per month

Reach out to Azure Support to request a limit increase.

For more information, please review the [Azure Quantum pricing page](#). Review the relevant provider pricing pages in the Azure portal for details on third-party offerings.

<sup>1</sup> Describes the number of jobs that can be queued at the same time.

## Azure RBAC limits

The following limits apply to [Azure role-based access control \(Azure RBAC\)](#).

RESOURCE	LIMIT
Azure role assignments per Azure subscription	2,000
Azure role assignments per management group	500
Size of description for Azure role assignments	2 KB
Size of condition for Azure role assignments	8 KB
Azure custom roles per tenant	5,000
Azure custom roles per tenant (for Azure Germany and Azure China 21Vianet)	2,000

## Azure SignalR Service limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure SignalR Service units per instance for Free tier	1	1
Azure SignalR Service units per instance for Standard tier	100	100
Azure SignalR Service units per subscription per region for Free tier	5	5
Total Azure SignalR Service unit counts per subscription per region	150	Unlimited
Connections per unit per day for Free tier	20	20
Connections per unit per day for Standard tier	1,000	1,000
Included messages per unit per day for Free tier	20,000	20,000
Additional messages per unit per day for Free tier	0	0
Included messages per unit per day for Standard tier	1,000,000	1,000,000
Additional messages per unit per day for Standard tier	Unlimited	Unlimited

To request an update to your subscription's default limits, open a support ticket.

For more information about how connections and messages are counted, see [Messages and connections in Azure SignalR Service](#).

If your requirements exceed the limits, switch from Free tier to Standard tier and add units. For more information, see [How to scale an Azure SignalR Service instance?](#).

If your requirements exceed the limits of a single instance, add instances. For more information, see [How to scale SignalR Service with multiple instances?](#).

## Azure VMware Solution limits

The following table describes the maximum limits for Azure VMware Solution.

RESOURCE	LIMIT
Clusters per private cloud	12
Minimum number of hosts per cluster	3

RESOURCE	LIMIT
Maximum number of hosts per cluster	16
hosts per private cloud	96
vCenter per private cloud	1
HCX site pairings	25 (any edition)
Azure VMware Solution ExpressRoute max linked private clouds	4 The virtual network gateway used determines the actual max linked private clouds. For more details, see <a href="#">About ExpressRoute virtual network gateways</a>
Azure VMware Solution ExpressRoute port speed	10 Gbps The virtual network gateway used determines the actual bandwidth. For more details, see <a href="#">About ExpressRoute virtual network gateways</a>
Public IPs exposed via vWAN	100
vSAN capacity limits	75% of total usable (keep 25% available for SLA)

For other VMware-specific limits, use the [VMware configuration maximum tool](#)!.

## Backup limits

For a summary of Azure Backup support settings and limitations, see [Azure Backup Support Matrices](#).

## Batch limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure Batch accounts per region per subscription	1-3	50
Dedicated cores per Batch account	90-900	Contact support
Low-priority cores per Batch account	10-100	Contact support
<b>Active</b> jobs and job schedules per Batch account ( <b>completed</b> jobs have no limit)	100-300	1,000 <sup>1</sup>
Pools per Batch account	20-100	500 <sup>1</sup>

<sup>1</sup>To request an increase beyond this limit, contact Azure Support.

### NOTE

Default limits vary depending on the type of subscription you use to create a Batch account. Cores quotas shown are for Batch accounts in Batch service mode. [View the quotas in your Batch account](#).

**IMPORTANT**

To help us better manage capacity during the global health pandemic, the default core quotas for new Batch accounts in some regions and for some types of subscription have been reduced from the above range of values, in some cases to zero cores. When you create a new Batch account, [check your core quota](#) and [request a core quota increase](#), if required. Alternatively, consider reusing Batch accounts that already have sufficient quota.

## Classic deployment model limits

If you use classic deployment model instead of the Azure Resource Manager deployment model, the following limits apply.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
vCPUs per <a href="#">subscription</a> <sup>1</sup>	20	10,000
<a href="#">Coadministrators</a> per subscription	200	200
<a href="#">Storage accounts</a> per subscription <sup>2</sup>	100	100
<a href="#">Cloud services</a> per subscription	20	200
<a href="#">Local networks</a> per subscription	10	500
DNS servers per subscription	9	100
Reserved IPs per subscription	20	100
<a href="#">Affinity groups</a> per subscription	256	256
Subscription name length (characters)	64	64

<sup>1</sup>Extra small instances count as one vCPU toward the vCPU limit despite using a partial CPU core.

<sup>2</sup>The storage account limit includes both Standard and Premium storage accounts.

## Container Instances limits

RESOURCE	LIMIT
Standard sku container groups per region per subscription	100 <sup>1</sup>
Dedicated sku container groups per region per subscription	0 <sup>1</sup>
Number of containers per container group	60
Number of volumes per container group	20
Standard sku cores (CPUs) per region per subscription	10 <sup>1,2</sup>
Standard sku cores (CPUs) for K80 GPU per region per subscription	18 <sup>1,2</sup>

RESOURCE	LIMIT
Standard sku cores (CPUs) for P100 or V100 GPU per region per subscription	0 <sup>1,2</sup>
Ports per IP	5
Container instance log size - running instance	4 MB
Container instance log size - stopped instance	16 KB or 1,000 lines
Container group creates per hour	300 <sup>1</sup>
Container group creates per 5 minutes	100 <sup>1</sup>
Container group deletes per hour	300 <sup>1</sup>
Container group deletes per 5 minutes	100 <sup>1</sup>

<sup>1</sup>To request a limit increase, create an [Azure Support request](#). Free subscriptions including [Azure Free Account](#) and [Azure for Students](#) aren't eligible for limit or quota increases. If you have a free subscription, you can [upgrade](#) to a Pay-As-You-Go subscription.

<sup>2</sup>Default limit for [Pay-As-You-Go](#) subscription. Limit may differ for other category types.

## Container Registry limits

The following table details the features and limits of the Basic, Standard, and Premium [service tiers](#).

RESOURCE	BASIC	STANDARD	PREMIUM
Included storage <sup>1</sup> (GiB)	10	100	500
Storage limit (TiB)	20	20	20
Maximum image layer size (GiB)	200	200	200
Maximum manifest size (MiB)	4	4	4
ReadOps per minute <sup>2, 3</sup>	1,000	3,000	10,000
WriteOps per minute <sup>2, 4</sup>	100	500	2,000
Download bandwidth <sup>2</sup> (Mbps)	30	60	100
Upload bandwidth <sup>2</sup> (Mbps)	10	20	50
Webhooks	2	10	500
Geo-replication	N/A	N/A	<a href="#">Supported</a>

RESOURCE	BASIC	STANDARD	PREMIUM
Availability zones	N/A	N/A	Preview
Content trust	N/A	N/A	Supported
Private link with private endpoints	N/A	N/A	Supported
• Private endpoints	N/A	N/A	200
Public IP network rules	N/A	N/A	100
Service endpoint VNet access	N/A	N/A	Preview
• Virtual network rules	N/A	N/A	100
Customer-managed keys	N/A	N/A	Supported
Repository-scoped permissions	N/A	N/A	Preview
• Tokens	N/A	N/A	20,000
• Scope maps	N/A	N/A	20,000
• Repositories per scope map	N/A	N/A	500

<sup>1</sup> Storage included in the daily rate for each tier. Additional storage may be used, up to the registry storage limit, at an additional daily rate per GiB. For rate information, see [Azure Container Registry pricing](#). If you need storage beyond the registry storage limit, please contact Azure Support.

<sup>2</sup> *ReadOps*, *WriteOps*, and *Bandwidth* are minimum estimates. Azure Container Registry strives to improve performance as usage requires.

<sup>3</sup> A `docker pull` translates to multiple read operations based on the number of layers in the image, plus the manifest retrieval.

<sup>4</sup> A `docker push` translates to multiple write operations, based on the number of layers that must be pushed. A `docker push` includes *ReadOps* to retrieve a manifest for an existing image.

## Content Delivery Network limits

RESOURCE	LIMIT
Azure Content Delivery Network profiles	25
Content Delivery Network endpoints per profile	25
Custom domains per endpoint	25
Maximum origin group per profile	10

RESOURCE	LIMIT
Maximum origin per origin group	10
Maximum number of rules per CDN endpoint	25
Maximum number of match conditions per rule	10
Maximum number of actions per rule	5

A Content Delivery Network subscription can contain one or more Content Delivery Network profiles. A Content Delivery Network profile can contain one or more Content Delivery Network endpoints. You might want to use multiple profiles to organize your Content Delivery Network endpoints by internet domain, web application, or some other criteria.

## Data Lake Analytics limits

Azure Data Lake Analytics makes the complex task of managing distributed infrastructure and complex code easy. It dynamically provisions resources, and you can use it to do analytics on exabytes of data. When the job completes, it winds down resources automatically. You pay only for the processing power that was used. As you increase or decrease the size of data stored or the amount of compute used, you don't have to rewrite code. To raise the default limits for your subscription, contact support.

RESOURCE	LIMIT	COMMENTS
Maximum number of concurrent jobs	20	
Maximum number of analytics units (AUs) per account	250	Use any combination of up to a maximum of 250 AUs across 20 jobs. To increase this limit, contact Microsoft Support.
Maximum script size for job submission	3 MB	
Maximum number of Data Lake Analytics accounts per region per subscription	5	To increase this limit, contact Microsoft Support.

## Data Factory limits

Azure Data Factory is a multitenant service that has the following default limits in place to make sure customer subscriptions are protected from each other's workloads. To raise the limits up to the maximum for your subscription, contact support.

### Version 2

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a data factory	5,000	<a href="#">Contact support</a> .

Resource	Default Limit	Maximum Limit
Total CPU cores for Azure-SSIS Integration Runtimes under one subscription	256	Contact support.
Concurrent pipeline runs per data factory that's shared among all pipelines in the factory	10,000	10,000
Concurrent External activity runs per subscription per <a href="#">Azure Integration Runtime region</a>  External activities are managed on integration runtime but execute on linked services, including Databricks, stored procedure, Web, and others. This limit does not apply to Self-hosted IR.	3,000	3,000
Concurrent Pipeline activity runs per subscription per <a href="#">Azure Integration Runtime region</a>  Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete. This limit does not apply to Self-hosted IR.	1,000	1,000
Concurrent authoring operations per subscription per <a href="#">Azure Integration Runtime region</a>  Including test connection, browse folder list and table list, preview data. This limit does not apply to Self-hosted IR.	200	200
Concurrent Data Integration Units <sup>1</sup> consumption per subscription per <a href="#">Azure Integration Runtime region</a>	Region group 1 <sup>2</sup> : 6,000 Region group 2 <sup>2</sup> : 3,000 Region group 3 <sup>2</sup> : 1,500 Managed virtual network <sup>2</sup> : 2,400	Region group 1 <sup>2</sup> : 6,000 Region group 2 <sup>2</sup> : 3,000 Region group 3 <sup>2</sup> : 1,500 Managed virtual network: Contact support.
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	Contact support.
Maximum parameters per pipeline	50	50
ForEach items	100,000	100,000
ForEach parallelism	20	50
Maximum queued runs per pipeline	100	100
Characters per expression	8,192	8,192

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Minimum tumbling window trigger interval	5 min	15 min
Maximum timeout for pipeline activity runs	7 days	7 days
Bytes per object for pipeline objects <sup>3</sup>	200 KB	200 KB
Bytes per object for dataset and linked service objects <sup>3</sup>	100 KB	2,000 KB
Bytes per payload for each activity run <sup>4</sup>	896 KB	896 KB
Data Integration Units <sup>1</sup> per copy activity run	256	256
Write API calls	1,200/h	1,200/h  This limit is imposed by Azure Resource Manager, not Azure Data Factory.
Read API calls	12,500/h	12,500/h  This limit is imposed by Azure Resource Manager, not Azure Data Factory.
Monitoring queries per minute	1,000	1,000
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	Contact support.
Concurrent number of data flows per integration runtime in managed vNet	20	Contact support.
Concurrent number of data flow debug sessions per user per factory	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a factory	2 GB	Contact support.

<sup>1</sup> The data integration unit (DIU) is used in a cloud-to-cloud copy operation, learn more from [Data integration units \(version 2\)](#). For information on billing, see [Azure Data Factory pricing](#).

<sup>2</sup> [Azure Integration Runtime](#) is [globally available](#) to ensure data compliance, efficiency, and reduced network egress costs.

Region group	Regions
Region group 1	Central US, East US, East US 2, North Europe, West Europe, West US, West US 2
Region group 2	Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US
Region group 3	Other regions

If managed virtual network is enabled, the data integration unit (DIU) in all region groups are 2,400.

<sup>3</sup> Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

<sup>4</sup> The payload for each activity run includes the activity configuration, the associated dataset(s) and linked service(s) configurations if any, and a small portion of system properties generated per activity type. Limit for this payload size doesn't relate to the amount of data you can move and process with Azure Data Factory. Learn about the [symptoms and recommendation](#) if you hit this limit.

## Version 1

Resource	Default limit	Maximum limit
Pipelines within a data factory	2,500	<a href="#">Contact support</a> .
Data sets within a data factory	5,000	<a href="#">Contact support</a> .
Concurrent slices per data set	10	10
Bytes per object for pipeline objects <sup>1</sup>	200 KB	200 KB
Bytes per object for data set and linked service objects <sup>1</sup>	100 KB	2,000 KB
Azure HDInsight on-demand cluster cores within a subscription <sup>2</sup>	60	<a href="#">Contact support</a> .
Cloud data movement units per copy activity run <sup>3</sup>	32	32
Retry count for pipeline activity runs	1,000	MaxInt (32 bit)

<sup>1</sup> Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Data Factory. Data Factory is designed to scale to handle petabytes of data.

<sup>2</sup> On-demand HDInsight cores are allocated out of the subscription that contains the data factory. As a result, the previous limit is the Data Factory-enforced core limit for on-demand HDInsight cores. It's different from the core limit that's associated with your Azure subscription.

<sup>3</sup> The cloud data movement unit (DMU) for version 1 is used in a cloud-to-cloud copy operation, learn more from [Cloud data movement units \(version 1\)](#). For information on billing, see [Azure Data Factory pricing](#).

RESOURCE	DEFAULT LOWER LIMIT	MINIMUM LIMIT
Scheduling interval	15 minutes	15 minutes
Interval between retry attempts	1 second	1 second
Retry timeout value	1 second	1 second

#### Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource Manager API limits](#).

## Data Lake Storage limits

Azure Data Lake Storage Gen2 is not a dedicated service or storage account type. It is the latest release of capabilities that are dedicated to big data analytics. These capabilities are available in a general-purpose v2 or BlockBlobStorage storage account, and you can obtain them by enabling the **Hierarchical namespace** feature of the account. For scale targets, see these articles.

- [Scale targets for Blob storage](#).
- [Scale targets for standard storage accounts](#).

Azure Data Lake Storage Gen1 is a dedicated service. It's an enterprise-wide hyper-scale repository for big data analytic workloads. You can use Data Lake Storage Gen1 to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics. There's no limit to the amount of data you can store in a Data Lake Storage Gen1 account.

RESOURCE	LIMIT	COMMENTS
Maximum number of Data Lake Storage Gen1 accounts, per subscription, per region	10	To request an increase for this limit, contact support.
Maximum number of access ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries.
Maximum number of default ACLs, per file or folder	32	This is a hard limit. Use groups to manage access with fewer entries.

## Data Share limits

Azure Data Share enables organizations to simply and securely share data with their customers and partners.

RESOURCE	LIMIT
Maximum number of Data Share resources per Azure subscription	100
Maximum number of sent shares per Data Share resource	200
Maximum number of received shares per Data Share resource	100
Maximum number of invitations per sent share	200

RESOURCE	LIMIT
Maximum number of share subscriptions per sent share	200
Maximum number of datasets per share	200
Maximum number of snapshot schedules per share	1

## Database Migration Service Limits

Azure Database Migration Service is a fully managed service designed to enable seamless migrations from multiple database sources to Azure data platforms with minimal downtime.

RESOURCE	LIMIT	COMMENTS
Maximum number of services per subscription, per region	10	To request an increase for this limit, contact support.

## Device Update for IoT Hub limits

### NOTE

When a given resource or operation doesn't have adjustable limits, the default and the maximum limits are the same. When the limit can be adjusted, the table includes different values for Default limit and Maximum limit headers. The limit can be raised above the default limit but not above the maximum limit. If you want to raise the limit or quota above the default limit, open an [online customer support request](#).

This table provides the limits for the Device Update for IoT Hub resource in Azure Resource Manager:

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT	ADJUSTABLE?
Accounts per subscription	2	25	Yes
Instances per account	2	25	Yes
Length of account name	Minimum: 3 Maximum: 24	Minimum: 3 Maximum: 24	No
Length of instance name	Minimum: 3 Maximum: 36	Minimum: 3 Maximum: 36	No

This table provides the various limits associated with the operations within Device Update for IoT Hub:

OPERATION	DEFAULT LIMIT	MAXIMUM LIMIT	ADJUSTABLE?
Number of devices per instance	10,000	10,000	No
Number of update providers per instance	25	25	No

OPERATION	DEFAULT LIMIT	MAXIMUM LIMIT	ADJUSTABLE?
Number of update names per provider per instance	25	25	No
Number of update versions per update provider and name per instance	100	100	No
Total number of updates per instance	100	100	No
Maximum single update file size	2 GB	2 GB	No
Maximum combined size of all files in a single import action	2 GB	2 GB	No
Number of device groups per instance	75	75	No

## Digital Twins limits

### NOTE

Some areas of this service have adjustable limits, and others do not. This is represented in the tables below with the *Adjustable?* column. When the limit can be adjusted, the *Adjustable?* value is *Yes*.

### Functional limits

The following table lists the functional limits of Azure Digital Twins.

### TIP

For modeling recommendations to operate within these functional limits, see [Modeling best practices](#).

AREA	CAPABILITY	DEFAULT LIMIT	ADJUSTABLE?
Azure resource	Number of Azure Digital Twins instances in a region, per subscription	10	Yes
Digital twins	Number of twins in an Azure Digital Twins instance	500,000	Yes
Digital twins	Number of incoming relationships to a single twin	5,000	No
Digital twins	Number of outgoing relationships from a single twin	5,000	No

Area	Capability	Default Limit	Adjustable?
Digital twins	Maximum size (of JSON body in a PUT or PATCH request) of a single twin	32 KB	No
Digital twins	Maximum request payload size	32 KB	No
Digital twins	Maximum size of a string property value (UTF-8)	4 KB	No
Digital twins	Maximum size of a property name	1 KB	No
Routing	Number of endpoints for a single Azure Digital Twins instance	6	No
Routing	Number of routes for a single Azure Digital Twins instance	6	Yes
Models	Number of models within a single Azure Digital Twins instance	10,000	Yes
Models	Number of models that can be uploaded in a single API call	250	No
Models	Maximum size (of JSON body in a PUT or PATCH request) of a single model	1 MB	No
Models	Number of items returned in a single page	100	No
Query	Number of items returned in a single page	100	Yes
Query	Number of AND / OR expressions in a query	50	Yes
Query	Number of array items in an IN / NOT IN clause	50	Yes
Query	Number of characters in a query	8,000	Yes
Query	Number of JOINS in a query	5	Yes

## Rate limits

The following table reflects the rate limits of different APIs.

API	Capability	Default Limit	Adjustable?
Models API	Number of requests per second	100	Yes
Digital Twins API	Number of read requests per second	1,000	Yes
Digital Twins API	Number of patch requests per second	1,000	Yes
Digital Twins API	Number of create/delete operations per second across all twins and relationships	50	Yes
Digital Twins API	Number of create/update/delete operations per second on a <b>single twin</b> or its relationships	10	No
Query API	Number of requests per second	500	Yes
Query API	Query Units per second	4,000	Yes
Event Routes API	Number of requests per second	100	Yes

### Other limits

Limits on data types and fields within DTDL documents for Azure Digital Twins models can be found within its spec documentation in GitHub: [Digital Twins Definition Language \(DTDL\) - version 2](#).

Query latency details are described in [Query language](#). Limitations of particular query language features can be found in the [query reference documentation](#).

## Event Grid limits

The following limits apply to Azure Event Grid **topics** (system, custom, and partner topics).

#### NOTE

These limits are per region.

Resource	Limit
Custom topics per Azure subscription	100
Event subscriptions per topic	500
Publish rate for a custom or a partner topic (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)
Event size	1 MB

RESOURCE	LIMIT
Private endpoint connections per topic	64
IP Firewall rules per topic	16

The following limits apply to Azure Event Grid **domains**.

RESOURCE	LIMIT
Topics per event domain	100,000
Event subscriptions per topic within a domain	500
Domain scope event subscriptions	50
Publish rate for an event domain (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)
Event Domains per Azure Subscription	100
Private endpoint connections per domain	64
IP Firewall rules per domain	16

## Event Hubs limits

The following tables provide quotas and limits specific to [Azure Event Hubs](#). For information about Event Hubs pricing, see [Event Hubs pricing](#).

### Common limits for all tiers

The following limits are common across all tiers.

LIMIT	NOTES	VALUE
Size of an event hub name	-	256 characters
Size of a consumer group name	Kafka protocol doesn't require the creation of a consumer group. AMQP: 50 characters	Kafka: 256 characters AMQP: 50 characters
Number of non-epoch receivers per consumer group	-	5
Number of authorization rules per namespace	Subsequent requests for authorization rule creation are rejected.	12
Number of calls to the GetRuntimeInformation method	-	50 per second
Number of virtual networks (VNet)	-	128
Number of IP Config rules	-	128

LIMIT	NOTES	VALUE
Maximum length of a schema group name		50
Maximum length of a schema name		100
Size in bytes per schema		1 MB
Number of properties per schema group		1024
Size in bytes per schema group property key		256
Size in bytes per schema group property value		1024

### Basic vs. standard vs. premium vs. dedicated tiers

The following table shows limits that may be different for basic, standard, and dedicated tiers. In the table CU is capacity unit, PU is processing unit, TU is throughput unit.

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Maximum size of Event Hubs publication	256 KB	1 MB	1 MB	1 MB
Number of consumer groups per event hub	1	20	100	1000 No limit per CU
Number of brokered connections per namespace	100	5,000	10000 per processing unit per PU	100, 000 per CU
Maximum retention period of event data	1 day	7 days	90 days 1 TB per PU	90 days 10 TB per CU
Maximum TUs or PUs or CUs	40 TUs	40 TUs	16 PUs	20 CUs
Number of partitions per event hub	32	32	100	1024 per event hub 2000 per CU
Number of namespaces per subscription	1000	1000	1000	1000 (50 per CU)
Number of event hubs per namespace	10	10	100 per PU	1000
Capture	N/A	Pay per hour	Included	Included

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Size of the schema registry (namespace) in mega bytes	N/A	25	100	1024
Number of schema groups in a schema registry or namespace	N/A	1 - excluding the default group	100 1 MB per schema	1000 1 MB per schema
Number of schema versions across all schema groups	N/A	25	1000	10000
Throughput per unit	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	No limits per PU *	No limits per CU *

\* Depends on various factors such as resource allocation, number of partitions, storage, and so on.

#### NOTE

You can publish events individually or batched. The publication limit (according to SKU) applies regardless of whether it is a single event or a batch. Publishing events larger than the maximum threshold will be rejected.

## IoT Central limits

IoT Central limits the number of applications you can deploy in a subscription to 10. If you need to increase this limit, contact [Microsoft support](#).

## IoT Hub limits

The following table lists the limits associated with the different service tiers S1, S2, S3, and F1. For information about the cost of each *unit* in each tier, see [Azure IoT Hub pricing](#).

RESOURCE	S1 STANDARD	S2 STANDARD	S3 STANDARD	F1 FREE
Messages/day	400,000	6,000,000	300,000,000	8,000
Maximum units	200	200	10	1

#### NOTE

If you anticipate using more than 200 units with an S1 or S2 tier hub or 10 units with an S3 tier hub, contact Microsoft Support.

The following table lists the limits that apply to IoT Hub resources.

RESOURCE	LIMIT
Maximum paid IoT hubs per Azure subscription	50
Maximum free IoT hubs per Azure subscription	1
Maximum number of characters in a device ID	128
Maximum number of device identities returned in a single call	1,000
IoT Hub message maximum retention for device-to-cloud messages	7 days
Maximum size of device-to-cloud message	256 KB
Maximum size of device-to-cloud batch	AMQP and HTTP: 256 KB for the entire batch MQTT: 256 KB for each message
Maximum messages in device-to-cloud batch	500
Maximum size of cloud-to-device message	64 KB
Maximum TTL for cloud-to-device messages	2 days
Maximum delivery count for cloud-to-device messages	100
Maximum cloud-to-device queue depth per device	50
Maximum delivery count for feedback messages in response to a cloud-to-device message	100
Maximum TTL for feedback messages in response to a cloud-to-device message	2 days
<a href="#">Maximum size of device twin</a>	8 KB for tags section, and 32 KB for desired and reported properties sections each
Maximum length of device twin string key	1 KB
Maximum length of device twin string value	4 KB
<a href="#">Maximum depth of object in device twin</a>	10
Maximum size of direct method payload	128 KB
Job history maximum retention	30 days
Maximum concurrent jobs	10 (for S3), 5 for (S2), 1 (for S1)
Maximum additional endpoints	10 (for S1, S2, and S3)

RESOURCE	LIMIT
Maximum message routing rules	100 (for S1, S2, and S3)
Maximum number of concurrently connected device streams	50 (for S1, S2, S3, and F1 only)
Maximum device stream data transfer	300 MB per day (for S1, S2, S3, and F1 only)

**NOTE**

If you need more than 50 paid IoT hubs in an Azure subscription, contact Microsoft Support.

**NOTE**

Currently, the total number of devices plus modules that can be registered to a single IoT hub is capped at 1,000,000. If you want to increase this limit, contact [Microsoft Support](#).

IoT Hub throttles requests when the following quotas are exceeded.

THROTTLE	PER-HUB VALUE
Identity registry operations (create, retrieve, list, update, and delete), individual or bulk import/export	83.33/sec/unit (5,000/min/unit) (for S3). 1.67/sec/unit (100/min/unit) (for S1 and S2).
Device connections	6,000/sec/unit (for S3), 120/sec/unit (for S2), 12/sec/unit (for S1). Minimum of 100/sec.
Device-to-cloud sends	6,000/sec/unit (for S3), 120/sec/unit (for S2), 12/sec/unit (for S1). Minimum of 100/sec.
Cloud-to-device sends	83.33/sec/unit (5,000/min/unit) (for S3), 1.67/sec/unit (100/min/unit) (for S1 and S2).
Cloud-to-device receives	833.33/sec/unit (50,000/min/unit) (for S3), 16.67/sec/unit (1,000/min/unit) (for S1 and S2).
File upload operations	83.33 file upload initiations/sec/unit (5,000/min/unit) (for S3), 1.67 file upload initiations/sec/unit (100/min/unit) (for S1 and S2). 10,000 SAS URIs can be out for an Azure Storage account at one time. 10 SAS URIs/device can be out at one time.
Direct methods	24 MB/sec/unit (for S3), 480 KB/sec/unit (for S2), 160 KB/sec/unit (for S1). Based on 8-KB throttling meter size.
Device twin reads	500/sec/unit (for S3), Maximum of 100/sec or 10/sec/unit (for S2), 100/sec (for S1)

THROTTLE	PER-HUB VALUE
Device twin updates	250/sec/unit (for S3), Maximum of 50/sec or 5/sec/unit (for S2), 50/sec (for S1)
Jobs operations (create, update, list, and delete)	83.33/sec/unit (5,000/min/unit) (for S3), 1.67/sec/unit (100/min/unit) (for S2), 1.67/sec/unit (100/min/unit) (for S1).
Jobs per-device operation throughput	50/sec/unit (for S3), maximum of 10/sec or 1/sec/unit (for S2), 10/sec (for S1).
Device stream initiation rate	5 new streams/sec (for S1, S2, S3, and F1 only).

## IoT Hub Device Provisioning Service limits

### NOTE

Some areas of this service have adjustable limits. This is represented in the tables below with the *Adjustable?* column. When the limit can be adjusted, the *Adjustable?* value is *Yes*.

The actual value to which a limit can be adjusted may vary based on each customer's deployment. Multiple instances of DPS may be required for very large deployments.

If your business requires raising an adjustable limit or quota above the default limit, you can request additional resources by [opening a support ticket](#).

The following table lists the limits that apply to Azure IoT Hub Device Provisioning Service resources.

RESOURCE	LIMIT	ADJUSTABLE?
Maximum device provisioning services per Azure subscription	10	Yes
Maximum number of registrations	1,000,000	Yes
Maximum number of individual enrollments	1,000,000	Yes
Maximum number of enrollment groups ( <i>X.509 certificate</i> )	100	Yes
Maximum number of enrollment groups ( <i>symmetric key</i> )	100	No
Maximum number of CAs	25	No
Maximum number of linked IoT hubs	50	No
Maximum size of message	96 KB	No

**TIP**

If the hard limit on symmetric key enrollment groups is a blocking issue, it is recommended to use individual enrollments as a workaround.

The Device Provisioning Service has the following rate limits.

RATE	PER-UNIT VALUE	ADJUSTABLE?
Operations	200/min/service	Yes
Device registrations	200/min/service	Yes
Device polling operation	5/10 sec/device	No

Each API call on DPS is billable as one *Operation*. This includes all the service APIs and the device registration API. The device registration polling operation is not billed.

## Key Vault limits

Azure Key Vault service supports two resource types: Vaults and Managed HSMs. The following two sections describe the service limits for each of them respectively.

### Resource type: vault

This section describes service limits for resource type `vaults`.

#### Key transactions (maximum transactions allowed in 10 seconds, per vault per region<sup>1</sup>):

KEY TYPE	HSM KEY CREATE KEY	HSM KEY ALL OTHER TRANSACTIONS	SOFTWARE KEY CREATE KEY	SOFTWARE KEY ALL OTHER TRANSACTIONS
RSA 2,048-bit	5	1,000	10	2,000
RSA 3,072-bit	5	250	10	500
RSA 4,096-bit	5	125	10	250
ECC P-256	5	1,000	10	2,000
ECC P-384	5	1,000	10	2,000
ECC P-521	5	1,000	10	2,000
ECC SECP256K1	5	1,000	10	2,000

**NOTE**

In the previous table, we see that for RSA 2,048-bit software keys, 2,000 GET transactions per 10 seconds are allowed. For RSA 2,048-bit HSM-keys, 1,000 GET transactions per 10 seconds are allowed.

The throttling thresholds are weighted, and enforcement is on their sum. For example, as shown in the previous table, when you perform GET operations on RSA HSM-keys, it's eight times more expensive to use 4,096-bit keys compared to 2,048-bit keys. That's because  $1,000/125 = 8$ .

In a given 10-second interval, an Azure Key Vault client can do *only one* of the following operations before it encounters a **429** throttling HTTP status code:

- 2,000 RSA 2,048-bit software-key GET transactions
- 1,000 RSA 2,048-bit HSM-key GET transactions
- 125 RSA 4,096-bit HSM-key GET transactions
- 124 RSA 4,096-bit HSM-key GET transactions and 8 RSA 2,048-bit HSM-key GET transactions

**Secrets, managed storage account keys, and vault transactions:**

TRANSACTIONS TYPE	MAXIMUM TRANSACTIONS ALLOWED IN 10 SECONDS, PER VAULT PER REGION <sup>1</sup>
All transactions	2,000

For information on how to handle throttling when these limits are exceeded, see [Azure Key Vault throttling guidance](#).

<sup>1</sup> A subscription-wide limit for all transaction types is five times per key vault limit. For example, HSM-other transactions per subscription are limited to 5,000 transactions in 10 seconds per subscription.

**Backup keys, secrets, certificates**

When you back up a key vault object, such as a secret, key, or certificate, the backup operation will download the object as an encrypted blob. This blob can't be decrypted outside of Azure. To get usable data from this blob, you must restore the blob into a key vault within the same Azure subscription and Azure geography.

TRANSACTIONS TYPE	MAXIMUM KEY VAULT OBJECT VERSIONS ALLOWED
Backup individual key, secret, certificate	500

**NOTE**

Attempting to backup a key, secret, or certificate object with more versions than above limit will result in an error. It is not possible to delete previous versions of a key, secret, or certificate.

**Limits on count of keys, secrets and certificates:**

Key Vault does not restrict the number of keys, secrets or certificates that can be stored in a vault. The transaction limits on the vault should be taken into account to ensure that operations are not throttled.

Key Vault does not restrict the number of versions on a secret, key or certificate, but storing a large number of versions (500+) can impact the performance of backup operations. See [Azure Key Vault Backup](#).

**Azure Private Link integration**

**NOTE**

The number of key vaults with private endpoints enabled per subscription is an adjustable limit. The limit shown below is the default limit. If you would like to request a limit increase for your service, please create a support request and it will be assessed on a case by case basis.

RESOURCE	LIMIT
Private endpoints per key vault	64
Key vaults with private endpoints per subscription	400

**Resource type: Managed HSM**

This section describes service limits for resource type `managed HSM`.

**Object limits**

ITEM	LIMITS
Number of HSM instances per subscription per region	5
Number of keys per HSM Pool	5000
Number of versions per key	100
Number of custom role definitions per HSM	50
Number of role assignments at HSM scope	50
Number of role assignment at each individual key scope	10

**Transaction limits for administrative operations (number of operations per second per HSM instance)**

OPERATION	NUMBER OF OPERATIONS PER SECOND
All RBAC operations (includes all CRUD operations for role definitions and role assignments)	5
Full HSM Backup/Restore (only one concurrent backup or restore operation per HSM instance supported)	1

**Transaction limits for cryptographic operations (number of operations per second per HSM instance)**

- Each Managed HSM instance constitutes 3 load balanced HSM partitions. The throughput limits are a function of underlying hardware capacity allocated for each partition. The tables below show maximum throughput with at least one partition available. Actual throughput may be up to 3x higher if all 3 partitions are available.
- Throughput limits noted assume that one single key is being used to achieve maximum throughput. For example, if a single RSA-2048 key is used the maximum throughput will be 1100 sign operations. If you use 1100 different keys with 1 transaction per second each, they will not be able to achieve the same throughput.

OPERATION	2048-BIT	3072-BIT	4096-BIT
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	10000	10000	6000
Decrypt	1100	360	160
Wrap	10000	10000	6000
Unwrap	1100	360	160
Sign	1100	360	160
Verify	10000	10000	6000

**EC key operations (number of operations per second per HSM instance)**

This table describes number of operations per second for each curve type.

OPERATION	P-256	P-256K	P-384	P-521
Create Key	1	1	1	1
Delete Key (soft-delete)	10	10	10	10
Purge Key	10	10	10	10
Backup Key	10	10	10	10
Restore Key	10	10	10	10
Get Key Information	1100	1100	1100	1100
Sign	260	260	165	56
Verify	130	130	82	28

**AES key operations (number of operations per second per HSM instance)**

- Encrypt and Decrypt operations assume a 4KB packet size.
- Throughput limits for Encrypt/Decrypt apply to AES-CBC and AES-GCM algorithms.

- Throughput limits for Wrap/Unwrap apply to AES-KW algorithm.

OPERATION	128-BIT	192-BIT	256-BIT
Create Key	1	1	1
Delete Key (soft-delete)	10	10	10
Purge Key	10	10	10
Backup Key	10	10	10
Restore Key	10	10	10
Get Key Information	1100	1100	1100
Encrypt	8000	8000	8000
Decrypt	8000	8000	8000
Wrap	9000	9000	9000
Unwrap	9000	9000	9000

## Managed identity limits

- Each managed identity counts towards the object quota limit in an Azure AD tenant as described in [Azure AD service limits and restrictions](#).
- The rate at which managed identities can be created have the following limits:
  - Per Azure AD Tenant per Azure region: 400 create operations per 20 seconds.
  - Per Azure Subscription per Azure region : 80 create operations per 20 seconds.
- The rate at which a user-assigned managed identity can be assigned with an Azure resource :
  - Per Azure AD Tenant per Azure region: 400 assignment operations per 20 seconds.
  - Per Azure Subscription per Azure region : 300 assignment operations per 20 seconds.

## Media Services limits

**NOTE**

For resources that aren't fixed, open a support ticket to ask for an increase in the quotas. Don't create additional Azure Media Services accounts in an attempt to obtain higher limits.

### Account limits

RESOURCE	DEFAULT LIMIT
Media Services accounts in a single subscription	100 (fixed)

### Asset limits

RESOURCE	DEFAULT LIMIT
Assets per Media Services account	1,000,000

## Storage (media) limits

RESOURCE	DEFAULT LIMIT
File size	In some scenarios, there is a limit on the maximum file size supported for processing in Media Services. <sup>(1)</sup>
Storage accounts	100 <sup>(2)</sup> (fixed)

<sup>1</sup> The maximum size supported for a single blob is currently up to 5 TB in Azure Blob Storage. Additional limits apply in Media Services based on the VM sizes that are used by the service. The size limit applies to the files that you upload and also the files that get generated as a result of Media Services processing (encoding or analyzing). If your source file is larger than 260-GB, your Job will likely fail.

<sup>2</sup> The storage accounts must be from the same Azure subscription.

## Jobs (encoding & analyzing) limits

RESOURCE	DEFAULT LIMIT
Jobs per Media Services account	500,000 <sup>(3)</sup> (fixed)
Job inputs per Job	50 (fixed)
Job outputs per Job	20 (fixed)
Transforms per Media Services account	100 (fixed)
Transform outputs in a Transform	20 (fixed)
Files per job input	10 (fixed)

<sup>3</sup> This number includes queued, finished, active, and canceled Jobs. It does not include deleted Jobs.

Any Job record in your account older than 90 days will be automatically deleted, even if the total number of records is below the maximum quota.

## Live streaming limits

RESOURCE	DEFAULT LIMIT
Live Events <sup>(4)</sup> per Media Services account	5
Live Outputs per Live Event	3 <sup>(5)</sup>
Max Live Output duration	Size of the DVR window

<sup>4</sup> For detailed information about Live Event limitations, see [Live Event types comparison and limitations](#).

<sup>5</sup> Live Outputs start on creation and stop when deleted.

## Packaging & delivery limits

RESOURCE	DEFAULT LIMIT
Streaming Endpoints (stopped or running) per Media Services account	2
Dynamic Manifest Filters	100
Streaming Policies	100 <sup>(6)</sup>
Unique Streaming Locators associated with an Asset at one time	100 <sup>(7)</sup> (fixed)

<sup>6</sup> When using a custom [Streaming Policy](#), you should design a limited set of such policies for your Media Service account, and re-use them for your StreamingLocators whenever the same encryption options and protocols are needed. You should not be creating a new Streaming Policy for each Streaming Locator.

<sup>7</sup> Streaming Locators are not designed for managing per-user access control. To give different access rights to individual users, use Digital Rights Management (DRM) solutions.

## Protection limits

RESOURCE	DEFAULT LIMIT
Options per Content Key Policy	30
Licenses per month for each of the DRM types on Media Services key delivery service per account	1,000,000

## Support ticket

For resources that are not fixed, you may ask for the quotas to be raised, by opening a [support ticket](#). Include detailed information in the request on the desired quota changes, use-case scenarios, and regions required. Do not create additional Azure Media Services accounts in an attempt to obtain higher limits.

## Media Services v2 (legacy)

For limits specific to Media Services v2 (legacy), see [Media Services v2 \(legacy\)](#)

## Mobile Services limits

TIER	FREE	BASIC	STANDARD
API calls	500,000	1.5 million per unit	15 million per unit
Active devices	500	Unlimited	Unlimited
Scale	N/A	Up to 6 units	Unlimited units
Push notifications	Azure Notification Hubs Free tier included, up to 1 million pushes	Notification Hubs Basic tier included, up to 10 million pushes	Notification Hubs Standard tier included, up to 10 million pushes
Real-time messaging/ Web Sockets	Limited	350 per mobile service	Unlimited
Offline synchronizations	Limited	Included	Included

TIER	FREE	BASIC	STANDARD
Scheduled jobs	Limited	Included	Included
Azure SQL Database (required) Standard rates apply for additional capacity	20 MB included	20 MB included	20 MB included
CPU capacity	60 minutes per day	Unlimited	Unlimited
Outbound data transfer	165 MB per day (daily rollover)	Included	Included

For more information on limits and pricing, see [Azure Mobile Services pricing](#).

## Multi-Factor Authentication limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum number of trusted IP addresses or ranges per subscription	0	50
Remember my devices, number of days	14	60
Maximum number of app passwords	0	No limit
Allow X attempts during MFA call	1	99
Two-way text message timeout seconds	60	600
Default one-time bypass seconds	300	1,800
Lock user account after X consecutive MFA denials	Not set	99
Reset account lockout counter after X minutes	Not set	9,999
Unlock account after X minutes	Not set	9,999

## Networking limits

### Networking limits - Azure Resource Manager

The following limits apply only for networking resources managed through **Azure Resource Manager** per region per subscription. Learn how to [view your current resource usage against your subscription limits](#).

**NOTE**

We recently increased all default limits to their maximum limits. If there's no maximum limit column, the resource doesn't have adjustable limits. If you had these limits increased by support in the past and don't see updated limits in the following tables, [open an online customer support request at no charge](#)

RESOURCE	LIMIT
Virtual networks	1,000
Subnets per virtual network	3,000
Virtual network peerings per virtual network	500
Virtual network gateways (VPN gateways) per virtual network	1
Virtual network gateways (ExpressRoute gateways) per virtual network	1
DNS servers per virtual network	20
Private IP addresses per virtual network	65,536
Private IP addresses per network interface	256
Private IP addresses per virtual machine	256
Public IP addresses per network interface	256
Public IP addresses per virtual machine	256
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500,000
Network interface cards	65,536
Network Security Groups	5,000
NSG rules per NSG	1,000
IP addresses and ranges specified for source or destination in a security group	4,000
Application security groups	3,000
Application security groups per IP configuration, per NIC	20
IP configurations per application security group	4,000
Application security groups that can be specified within all security rules of a network security group	100

RESOURCE	LIMIT
User-defined route tables	200
User-defined routes per route table	400
Point-to-site root certificates per Azure VPN Gateway	20
Point-to-site revoked client certificates per Azure VPN Gateway	300
Virtual network TAPs	100
Network interface TAP configurations per virtual network TAP	100

#### Public IP address limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Public IP addresses <sup>1,2</sup>	10 for Basic.	Contact support.
Static Public IP addresses <sup>1</sup>	10 for Basic.	Contact support.
Standard Public IP addresses <sup>1</sup>	10	Contact support.
Public IP addresses per Resource Group	800	Contact support.
Public IP Prefixes	limited by number of Standard Public IPs in a subscription	Contact support.
Public IP prefix length	/28	Contact support.

<sup>1</sup>Default limits for Public IP addresses vary by offer category type, such as Free Trial, Pay-As-You-Go, CSP. For example, the default for Enterprise Agreement subscriptions is 1000.

<sup>2</sup>Public IP addresses limit refers to the total amount of Public IP addresses, including Basic and Standard.

#### Load balancer limits

The following limits apply only for networking resources managed through Azure Resource Manager per region per subscription. Learn how to [view your current resource usage against your subscription limits](#).

#### Standard Load Balancer

RESOURCE	LIMIT
Load balancers	1,000
Rules (Load Balancer + Inbound NAT) per resource	1,500
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations	600

RESOURCE	LIMIT
Backend pool size	1,000 IP configurations, single virtual network
Backend resources per Load Balancer <sup>1</sup>	1,200
High-availability ports rule	1 per internal frontend
Outbound rules per Load Balancer	600
Load Balancers per VM	2 (1 Public and 1 internal)

<sup>1</sup> The limit is up to 1,200 resources, in any combination of standalone virtual machine resources, availability set resources, and virtual machine scale-set placement groups.

## Basic Load Balancer

RESOURCE	LIMIT
Load balancers	1,000
Rules per resource	250
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations <sup>2</sup>	200
Backend pool size	300 IP configurations, single availability set
Availability sets per Load Balancer	1
Load Balancers per VM	2 (1 Public and 1 internal)

<sup>2</sup> The limit for a single discrete resource in a backend pool (standalone virtual machine, availability set, or virtual machine scale-set placement group) is to have up to 250 Frontend IP configurations across a single Basic Public Load Balancer and Basic Internal Load Balancer.

The following limits apply only for networking resources managed through the **classic** deployment model per subscription. Learn how to [view your current resource usage against your subscription limits](#).

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Virtual networks	100	100
Local network sites	20	50
DNS servers per virtual network	20	20
Private IP addresses per virtual network	4,096	4,096
Concurrent TCP or UDP flows per NIC of a virtual machine or role instance	500,000, up to 1,000,000 for two or more NICs.	500,000, up to 1,000,000 for two or more NICs.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Network Security Groups (NSGs)	200	200
NSG rules per NSG	200	1,000
User-defined route tables	200	200
User-defined routes per route table	400	400
Public IP addresses (dynamic)	500	500
Reserved public IP addresses	500	500
Public IP per deployment	5	Contact support
Private IP (internal load balancing) per deployment	1	1
Endpoint access control lists (ACLs)	50	50

## ExpressRoute limits

RESOURCE	LIMIT
ExpressRoute circuits per subscription	50
ExpressRoute circuits per region per subscription, with Azure Resource Manager	10
Maximum number of IPv4 routes advertised to Azure private peering with ExpressRoute Standard	4,000
Maximum number of IPv4 routes advertised to Azure private peering with ExpressRoute Premium add-on	10,000
Maximum number of IPv6 routes advertised to Azure private peering with ExpressRoute Standard	100
Maximum number of IPv6 routes advertised to Azure private peering with ExpressRoute Premium add-on	100
Maximum number of IPv4 routes advertised from Azure private peering from the VNet address space for an ExpressRoute connection	1,000
Maximum number of IPv6 routes advertised from Azure private peering from the VNet address space for an ExpressRoute connection	1,000
Maximum number of routes advertised to Microsoft peering with ExpressRoute Standard	200
Maximum number of routes advertised to Microsoft peering with ExpressRoute Premium add-on	200

RESOURCE	LIMIT
Maximum number of ExpressRoute circuits linked to the same virtual network in the same peering location	4
Maximum number of ExpressRoute circuits linked to the same virtual network in different peering locations	16 (For more information, see <a href="#">Gateway SKU</a> .)
Number of virtual network links allowed per ExpressRoute circuit	See the <a href="#">Number of virtual networks per ExpressRoute circuit table</a> .

#### Number of virtual networks per ExpressRoute circuit

CIRCUIT SIZE	NUMBER OF VIRTUAL NETWORK LINKS FOR STANDARD	NUMBER OF VIRTUAL NETWORK LINKS WITH PREMIUM ADD-ON
50 Mbps	10	20
100 Mbps	10	25
200 Mbps	10	25
500 Mbps	10	40
1 Gbps	10	50
2 Gbps	10	60
5 Gbps	10	75
10 Gbps	10	100
40 Gbps*	10	100
100 Gbps*	10	100

\* 100 Gbps ExpressRoute Direct Only

#### NOTE

Global Reach connections count against the limit of virtual network connections per ExpressRoute Circuit. For example, a 10 Gbps Premium Circuit would allow for 5 Global Reach connections and 95 connections to the ExpressRoute Gateways or 95 Global Reach connections and 5 connections to the ExpressRoute Gateways or any other combination up to the limit of 100 connections for the circuit.

#### Virtual Network Gateway limits

RESOURCE	LIMIT
VNet Address Prefixes	600 per VPN gateway
Aggregate BGP routes	4,000 per VPN gateway
Local Network Gateway address prefixes	1000 per local network gateway

RESOURCE	LIMIT
S2S connections	Depends on the gateway SKU
P2S connections	Depends on the gateway SKU
P2S route limit - IKEv2	256 for non-Windows / 25 for Windows
P2S route limit - OpenVPN	1000
Max. flows	100K for VpnGw1/AZ / 512K for VpnGw2-4/AZ

### NAT Gateway limits

RESOURCE	LIMIT
Public IP addresses	16 per NAT gateway

### Virtual WAN limits

RESOURCE	LIMIT
VPN (branch) connections per hub	1,000
Aggregate throughput per Virtual WAN Site-to-site VPN gateway	20 Gbps
Throughput per Virtual WAN VPN connection (2 tunnels)	2 Gbps with 1 Gbps/IPsec tunnel
Point-to-Site users per hub	100,000
Aggregate throughput per Virtual WAN User VPN (Point-to-site) gateway	200 Gbps
Aggregate throughput per Virtual WAN ExpressRoute gateway	20 Gbps
ExpressRoute Circuit connections per hub	8
VNet connections per hub	500 minus total number of hubs in Virtual WAN
Aggregate throughput per Virtual WAN Hub Router	50 Gbps for VNet to VNet transit
VM workload across all VNets connected to a single Virtual WAN hub	2000 (If you want to raise the limit or quota above the default limit, open an online customer support request.)

### Application Gateway limits

The following table applies to v1, v2, Standard, and WAF SKUs unless otherwise stated.

RESOURCE	LIMIT	NOTE
Azure Application Gateway	1,000 per subscription	

RESOURCE	LIMIT	NOTE
Front-end IP configurations	2	1 public and 1 private
Front-end ports	100 <sup>1</sup>	
Back-end address pools	100 <sup>1</sup>	
Back-end servers per pool	1,200	
HTTP listeners	200 <sup>1</sup>	<p>Limited to 100 active listeners that are routing traffic. Active listeners = total number of listeners - listeners not active.</p> <p>If a default configuration inside a routing rule is set to route traffic (for example, it has a listener, a backend pool, and HTTP settings) then that also counts as a listener. See <a href="#">Frequently asked questions about Application Gateway</a> for additional details.</p>
HTTP load-balancing rules	400 <sup>1</sup>	
Back-end HTTP settings	100 <sup>1</sup>	
Instances per gateway	V1 SKU - 32 V2 SKU - 125	
SSL certificates	100 <sup>1</sup>	1 per HTTP listener
Maximum SSL certificate size	V1 SKU - 10 KB V2 SKU - 16 KB	
Authentication certificates	100	
Trusted root certificates	100	
Request timeout minimum	1 second	
Request timeout maximum	24 hours	
Number of sites	100 <sup>1</sup>	1 per HTTP listener
URL maps per listener	1	
Maximum path-based rules per URL map	100	
Redirect configurations	100 <sup>1</sup>	
Number of rewrite rule sets	400	
Number of Header or URL configuration per rewrite rule set	40	

RESOURCE	LIMIT	NOTE
Number of conditions per rewrite rule set	40	
Concurrent WebSocket connections	Medium gateways 20k <sup>2</sup> Large gateways 50k <sup>2</sup>	
Maximum URL length	32KB	
Maximum header size for HTTP/2	16KB	
Maximum file upload size (Standard SKU)	V2 - 4 GB V1 - 2GB	
Maximum file upload size (WAF SKU)	V1 Medium - 100 MB V1 Large - 500 MB V2 - 750 MB V2 (with CRS 3.2 or newer) - 4GB	
WAF body size limit (without files)	V1 or V2 (with CRS 3.1 and older) - 128KB V2 (with CRS 3.2 or newer) - 2MB	
Maximum WAF custom rules	100	
Maximum WAF exclusions per Application Gateway	40	

<sup>1</sup> In case of WAF-enabled SKUs, you must limit the number of resources to 40.

<sup>2</sup> Limit is per Application Gateway instance not per Application Gateway resource.

## Network Watcher limits

RESOURCE	LIMIT	NOTE
Azure Network Watcher	1 per region	Network Watcher is created to enable access to the service. Only one instance of Network Watcher is required per subscription per region.
Packet capture sessions	10,000 per region	Number of sessions only, not saved captures.

## Private Link limits

The following limits apply to Azure private link:

RESOURCE	LIMIT
Number of private endpoints per virtual network	1000
Number of private endpoints per subscription	64000
Number of private link services per subscription	800

RESOURCE	LIMIT
Number of IP Configurations on a private link service	8 (This number is for the NAT IP addresses used per PLS)
Number of private endpoints on the same private link service	1000
Number of private endpoints per key vault	64
Number of key vaults with private endpoints per subscription	400
Number of private DNS zone groups that can be linked to a private endpoint	1
Number of DNS zones in each group	5

### Traffic Manager limits

RESOURCE	LIMIT
Profiles per subscription	200
Endpoints per profile	200

### Azure Bastion limits

WORKLOAD TYPE*	LIMIT**
Light	100
Medium	50
Heavy	5

\*These workload types are defined here: [Remote Desktop workloads](#)

\*\*These limits are based on RDP performance tests for Azure Bastion. The numbers may vary due to other on-going RDP sessions or other on-going SSH sessions.

### Azure DNS limits

#### Public DNS zones

RESOURCE	LIMIT
Public DNS Zones per subscription	250 <sup>1</sup>
Record sets per public DNS zone	10,000 <sup>1</sup>
Records per record set in public DNS zone	20
Number of Alias records for a single Azure resource	20

<sup>1</sup>If you need to increase these limits, contact Azure Support.

## Private DNS zones

RESOURCE	LIMIT
Private DNS zones per subscription	1000
Record sets per private DNS zone	25000
Records per record set for private DNS zones	20
Virtual Network Links per private DNS zone	1000
Virtual Networks Links per private DNS zones with auto-registration enabled	100
Number of private DNS zones a virtual network can get linked to with auto-registration enabled	1
Number of private DNS zones a virtual network can get linked	1000
Number of DNS queries a virtual machine can send to Azure DNS resolver, per second	1000 <sup>1</sup>
Maximum number of DNS queries queued (pending response) per virtual machine	200 <sup>1</sup>

<sup>1</sup>These limits are applied to every individual virtual machine and not at the virtual network level. DNS queries exceeding these limits are dropped.

## Azure Firewall limits

RESOURCE	LIMIT
Data throughput	30 Gbps
Rule limits	10,000 unique source/destinations in network and application rules
Total size of rules within a single Rule Collection Group	2 Mb
Number of Rule Collection Groups in a Firewall Policy	50
Maximum DNAT rules	298 (for firewalls configured with a single Public IP address)  The DNAT limitation is due to the underlying platform. The maximum number of DNAT rules is 298. However, any additional public IP addresses reduce the number of the available DNAT rules. For example, two public IP addresses allow for 297 DNAT rules. If a rule's protocol is configured for both TCP and UDP, it counts as two rules.
Minimum AzureFirewallSubnet size	/26
Port range in network and application rules	1 - 65535

RESOURCE	LIMIT
Public IP addresses	250 maximum. All public IP addresses can be used in DNAT rules and they all contribute to available SNAT ports.
IP addresses in IP Groups	Maximum of 100 IP Groups per firewall. Maximum 5000 individual IP addresses or IP prefixes per each IP Group.
Route table	<p>By default, AzureFirewallSubnet has a 0.0.0.0/0 route with the <b>NextHopType</b> value set to <b>Internet</b>.</p> <p>Azure Firewall must have direct Internet connectivity. If your AzureFirewallSubnet learns a default route to your on-premises network via BGP, you must override that with a 0.0.0.0/0 UDR with the <b>NextHopType</b> value set as <b>Internet</b> to maintain direct Internet connectivity. By default, Azure Firewall doesn't support forced tunneling to an on-premises network.</p> <p>However, if your configuration requires forced tunneling to an on-premises network, Microsoft will support it on a case by case basis. Contact Support so that we can review your case. If accepted, we'll allow your subscription and ensure the required firewall Internet connectivity is maintained.</p>
FQDNs in network rules	For good performance, do not exceed more than 1000 FQDNs across all network rules per firewall.

### Azure Front Door Service limits

RESOURCE	LIMIT
Azure Front Door resources per subscription	100
Front-end hosts, which includes custom domains per resource	500
Routing rules per resource	500
Back-end pools per resource	50
Back ends per back-end pool	100
Path patterns to match for a routing rule	25
URLs in a single cache purge call	100
Custom web application firewall rules per policy	100
Web application firewall policy per subscription	100
Web application firewall match conditions per custom rule	10
Web application firewall IP address ranges per custom rule	600

RESOURCE	LIMIT
Web application firewall string match values per match condition	10
Web application firewall string match value length	256
Web application firewall POST body parameter name length	256
Web application firewall HTTP header name length	256
Web application firewall cookie name length	256
Web application firewall exclusion limit	100
Web application firewall HTTP request body size inspected	128 KB
Web application firewall custom response body length	2 KB

### Azure Front Door Standard/Premium (Preview) Service Limits

\*\*\* Maximum 500 total Standard and Premium profiles per subscription.

RESOURCE	STANDARD SKU LIMIT	PREMIUM SKU LIMIT
Maximum endpoint per profile	10	25
Maximum custom domain per profile	100	200
Maximum origin group per profile	100	200
Maximum secrets per profile	100	200
Maximum security policy per profile	100	200
Maximum rule set per profile	100	200
Maximum rules per rule set	100	100
Maximum origin per origin group	50	50
Maximum routes per endpoint	100	200
URLs in a single cache purge call	100	100
Custom web application firewall rules per policy	100	100
Web application firewall match conditions per custom rule	10	10
Web application firewall IP address ranges per custom rule	600	600

RESOURCE	STANDARD SKU LIMIT	PREMIUM SKU LIMIT
Web application firewall string match values per match condition	10	10
Web application firewall string match value length	256	256
Web application firewall POST body parameter name length	256	256
Web application firewall HTTP header name length	256	256
Web application firewall cookie name length	256	256
Web application firewall HTTP request body size inspected	128 KB	128 KB
Web application firewall custom response body length	2 KB	2 KB

#### Timeout values

##### Client to Front Door

- Front Door has an idle TCP connection timeout of 61 seconds.

##### Front Door to application back-end

- If the response is a chunked response, a 200 is returned if or when the first chunk is received.
- After the HTTP request is forwarded to the back end, Front Door waits for 30 seconds for the first packet from the back end. Then it returns a 503 error to the client. This value is configurable via the field `sendRecvTimeoutSeconds` in the API.
  - If a request is cached and it takes more than 30 seconds for the first packet from Front Door or from the backend, then a 504 error is returned to the client.
- After the first packet is received from the back end, Front Door waits for 30 seconds in an idle timeout. Then it returns a 503 error to the client. This timeout value is not configurable.
- Front Door to the back-end TCP session timeout is 90 seconds.

#### Upload and download data limit

	WITH CHUNKED TRANSFER ENCODING (CTE)	WITHOUT HTTP CHUNKING
Download	There's no limit on the download size.	There's no limit on the download size.
Upload	There's no limit as long as each CTE upload is less than 2 GB.	The size can't be larger than 2 GB.

#### Other limits

- Maximum URL size - 8,192 bytes - Specifies maximum length of the raw URL (scheme + hostname + port + path + query string of the URL)
- Maximum Query String size - 4,096 bytes - Specifies the maximum length of the query string, in bytes.
- Maximum HTTP response header size from health probe URL - 4,096 bytes - Specified the maximum length of all the response headers of health probes.
- Maximum rules engine action header value character: 640 characters.

- Maximum rules engine condition header value character: 256 characters.
- Maximum ETag header size: 128 bytes

For more information about limits that apply to Rules Engine configurations, see [Rules Engine terminology](#)

## Notification Hubs limits

TIER	FREE	BASIC	STANDARD
Included pushes	1 million	10 million	10 million
Active devices	500	200,000	10 million
Tag quota per installation or registration	60	60	60

For more information on limits and pricing, see [Notification Hubs pricing](#).

## Purview limits

The latest values for Azure Purview quotas can be found in the [Azure Purview quota page](#).

## Service Bus limits

The following table lists quota information specific to Azure Service Bus messaging. For information about pricing and other quotas for Service Bus, see [Service Bus pricing](#).

QUOTA NAME	SCOPE	VALUE	NOTES
Maximum number of namespaces per Azure subscription	Namespace	1000 (default and maximum)	Subsequent requests for additional namespaces are rejected.
Queue or topic size	Entity	<p>1, 2, 3, 4 GB or 5 GB</p> <p>In the Premium SKU, and the Standard SKU with <a href="#">partitioning</a> enabled, the maximum queue or topic size is 80 GB.</p> <p>Total size limit for a premium namespace is 1 TB per <a href="#">messaging unit</a>. Total size of all entities in a namespace can't exceed this limit.</p>	<p>Defined upon creation/updation of the queue or topic.</p> <p>Subsequent incoming messages are rejected, and an exception is received by the calling code.</p>
Number of concurrent connections on a namespace	Namespace	<p>Net Messaging: 1,000.</p> <p>AMQP: 5,000.</p>	Subsequent requests for additional connections are rejected, and an exception is received by the calling code. REST operations don't count toward concurrent TCP connections.

Quota name	Scope	Value	Notes
Number of concurrent receive requests on a queue, topic, or subscription entity	Entity	5,000	Subsequent receive requests are rejected, and an exception is received by the calling code. This quota applies to the combined number of concurrent receive operations across all subscriptions on a topic.
Number of topics or queues per namespace	Namespace	10,000 for the Basic or Standard tier. The total number of topics and queues in a namespace must be less than or equal to 10,000.  For the Premium tier, 1,000 per messaging unit (MU).	Subsequent requests for creation of a new topic or queue on the namespace are rejected. As a result, if configured through the <a href="#">Azure portal</a> , an error message is generated. If called from the management API, an exception is received by the calling code.
Number of <a href="#">partitioned topics or queues</a> per namespace	Namespace	Basic and Standard tiers: 100.  Partitioned entities aren't supported in the Premium tier.  Each partitioned queue or topic counts toward the quota of 1,000 entities per namespace.	Subsequent requests for creation of a new partitioned topic or queue in the namespace are rejected. As a result, if configured through the <a href="#">Azure portal</a> , an error message is generated. If called from the management API, the exception <b>QuotaExceededException</b> is received by the calling code.  If you want to have more partitioned entities in a basic or a standard tier namespace, create additional namespaces.
Maximum size of any messaging entity path: queue or topic	Entity	-	260 characters.
Maximum size of any messaging entity name: namespace, subscription, or subscription rule	Entity	-	50 characters.
Maximum size of a message ID	Entity	-	128
Maximum size of a message session ID	Entity	-	128

Quota name	Scope	Value	Notes
Message size for a queue, topic, or subscription entity	Entity	Incoming messages that exceed these quotas are rejected, and an exception is received by the calling code.	256 KB for Standard tier 1 MB for Premium tier.  The message size includes the size of properties (system and user) and the size of payload. The size of system properties varies depending on your scenario.
Message property size for a queue, topic, or subscription entity	Entity	The exception <code>SerializationException</code> is generated.	Maximum message property size for each property is 32 KB.  Cumulative size of all properties can't exceed 64 KB. This limit applies to the entire header of the brokered message, which has both user properties and system properties, such as sequence number, label, and message ID.  Maximum number of header properties in property bag: <code>byte/int.MaxValue</code> .
Number of subscriptions per topic	Entity	Subsequent requests for creating additional subscriptions for the topic are rejected. As a result, if configured through the portal, an error message is shown. If called from the management API, an exception is received by the calling code.	2,000 per-topic for the Standard tier and Premium tier.
Number of SQL filters per topic	Entity	Subsequent requests for creation of additional filters on the topic are rejected, and an exception is received by the calling code.	2,000
Number of correlation filters per topic	Entity	Subsequent requests for creation of additional filters on the topic are rejected, and an exception is received by the calling code.	100,000

Quota name	Scope	Value	Notes
Size of SQL filters or actions	Namespace	Subsequent requests for creation of additional filters are rejected, and an exception is received by the calling code.	Maximum length of filter condition string: 1,024 (1 K). Maximum length of rule action string: 1,024 (1 K). Maximum number of expressions per rule action: 32.
Number of shared access authorization rules per namespace, queue, or topic	Entity, namespace	Subsequent requests for creation of additional rules are rejected, and an exception is received by the calling code.	Maximum number of rules per entity type: 12. Rules that are configured on a Service Bus namespace apply to all types: queues, topics.
Number of messages per transaction	Transaction	Additional incoming messages are rejected, and an exception stating "Can't send more than 100 messages in a single transaction" is received by the calling code.	100 For both <code>Send()</code> and <code>SendAsync()</code> operations.
Number of virtual network and IP filter rules	Namespace		128

## Site Recovery limits

The following limits apply to Azure Site Recovery.

Limit identifier	Limit
Number of vaults per subscription	500
Number of servers per Recovery Services vault	250
Number of protection groups per Recovery Services vault	No limit
Number of recovery plans per Recovery Services vault	No limit
Number of servers per protection group	No limit
Number of servers per recovery plan	100

## SQL Database limits

For SQL Database limits, see [SQL Database resource limits for single databases](#), [SQL Database resource limits for elastic pools and pooled databases](#), and [SQL Database resource limits for SQL Managed Instance](#).

The maximum number of private endpoints per Azure SQL Database logical server is 250.

# Azure Synapse Analytics limits

Azure Synapse Analytics has the following default limits to ensure customer's subscriptions are protected from each other's workloads. To raise the limits to the maximum for your subscription, contact support.

## Synapse Workspace Limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Synapse workspaces in an Azure subscription	20	20

## Synapse Pipeline Limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Synapse pipelines in a Synapse workspace	800	800
Total number of entities, such as pipelines, data sets, triggers, linked services, Private Endpoints, and integration runtimes, within a workspace	5,000	Contact support.
Total CPU cores for Azure-SSIS Integration Runtimes under one workspace	256	Contact support.
Concurrent pipeline runs per workspace that's shared among all pipelines in the workspace	10,000	10,000
Concurrent External activity runs per workspace per <a href="#">Azure Integration Runtime region</a>  External activities are managed on integration runtime but execute on linked services, including Databricks, stored procedure, HDInsight, Web, and others. This limit does not apply to Self-hosted IR.	3,000	3,000
Concurrent Pipeline activity runs per workspace per <a href="#">Azure Integration Runtime region</a>  Pipeline activities execute on integration runtime, including Lookup, GetMetadata, and Delete. This limit does not apply to Self-hosted IR.	1,000	1,000
Concurrent authoring operations per workspace per <a href="#">Azure Integration Runtime region</a>  Including test connection, browse folder list and table list, preview data. This limit does not apply to Self-hosted IR.	200	200

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Concurrent Data Integration Units <sup>1</sup> consumption per workspace per <a href="#">Azure Integration Runtime region</a>	Region group 1 <sup>2</sup> : 6,000 Region group 2 <sup>2</sup> : 3,000 Region group 3 <sup>2</sup> : 1,500 Managed virtual network <sup>2</sup> : 2,400	Region group 1 <sup>2</sup> : 6,000 Region group 2 <sup>2</sup> : 3,000 Region group 3 <sup>2</sup> : 1,500 Managed virtual network: <a href="#">Contact support</a> .
Maximum activities per pipeline, which includes inner activities for containers	40	40
Maximum number of linked integration runtimes that can be created against a single self-hosted integration runtime	100	<a href="#">Contact support</a> .
Maximum parameters per pipeline	50	50
ForEach items	100,000	100,000
ForEach parallelism	20	50
Maximum queued runs per pipeline	100	100
Characters per expression	8,192	8,192
Minimum tumbling window trigger interval	5 min	15 min
Maximum timeout for pipeline activity runs	7 days	7 days
Bytes per object for pipeline objects <sup>3</sup>	200 KB	200 KB
Bytes per object for dataset and linked service objects <sup>3</sup>	100 KB	2,000 KB
Bytes per payload for each activity run <sup>4</sup>	896 KB	896 KB
Data Integration Units <sup>1</sup> per copy activity run	256	256
Write API calls	1,200/h	1,200/h  This limit is imposed by Azure Resource Manager, not Azure Synapse Analytics.
Read API calls	12,500/h	12,500/h  This limit is imposed by Azure Resource Manager, not Azure Synapse Analytics.
Monitoring queries per minute	1,000	1,000

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per integration runtime	50	<a href="#">Contact support.</a>
Concurrent number of data flows per integration runtime in managed vNet	20	<a href="#">Contact support.</a>
Concurrent number of data flow debug sessions per user per workspace	3	3
Data Flow Azure IR TTL limit	4 hrs	4 hrs
Meta Data Entity Size limit in a workspace	2 GB	<a href="#">Contact support.</a>

<sup>1</sup> The data integration unit (DIU) is used in a cloud-to-cloud copy operation, learn more from [Data integration units \(version 2\)](#). For information on billing, see [Azure Synapse Analytics Pricing](#).

<sup>2</sup> [Azure Integration Runtime](#) is [globally available](#) to ensure data compliance, efficiency, and reduced network egress costs.

| Region group | Regions || ----- | ----- || Region group 1 | Central US, East US, East US 2, North Europe, West Europe, West US, West US 2 || Region group 2 | Australia East, Australia Southeast, Brazil South, Central India, Japan East, North Central US, South Central US, Southeast Asia, West Central US || Region group 3 | Other regions | If managed virtual network is enabled, the data integration unit (DIU) in all region groups are 2,400.

<sup>3</sup> Pipeline, data set, and linked service objects represent a logical grouping of your workload. Limits for these objects don't relate to the amount of data you can move and process with Azure Synapse Analytics. Synapse Analytics is designed to scale to handle petabytes of data.

<sup>4</sup> The payload for each activity run includes the activity configuration, the associated dataset(s) and linked service(s) configurations if any, and a small portion of system properties generated per activity type. Limit for this payload size doesn't relate to the amount of data you can move and process with Azure Synapse Analytics. Learn about the [symptoms and recommendation](#) if you hit this limit.

### Dedicated SQL pool limits

For details of capacity limits for dedicated SQL pools in Azure Synapse Analytics, see [dedicated SQL pool resource limits](#).

### Web service call limits

Azure Resource Manager has limits for API calls. You can make API calls at a rate within the [Azure Resource Manager API limits](#).

## Azure Files and Azure File Sync

To learn more about the limits for Azure Files and File Sync, see [Azure Files scalability and performance targets](#).

## Storage limits

The following table describes default limits for Azure general-purpose v2 (GPv2), general-purpose v1 (GPv1), and Blob storage accounts. The *ingress* limit refers to all data that is sent to a storage account. The *egress* limit

refers to all data that is received from a storage account.

Microsoft recommends that you use a GPv2 storage account for most scenarios. You can easily upgrade a GPv1 or a Blob storage account to a GPv2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a GPv2 storage account](#).

**NOTE**

You can request higher capacity and ingress limits. To request an increase, contact [Azure Support](#).

RESOURCE	LIMIT
Number of storage accounts per region per subscription, including standard, and premium storage accounts.	250
Default maximum storage account capacity	5 PiB <sup>1</sup>
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account.	No limit
Default maximum request rate per storage account	20,000 requests per second <sup>1</sup>
Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (LRS/GRS): <ul style="list-style-type: none"><li>• Australia East</li><li>• Central US</li><li>• East Asia</li><li>• East US 2</li><li>• France Central</li><li>• Japan East</li><li>• Korea Central</li><li>• North Europe</li><li>• South Central US</li><li>• Southeast Asia</li><li>• UK South</li><li>• West Europe</li><li>• West US</li></ul>	60 Gbps <sup>1</sup>
Default maximum ingress per general-purpose v2 and Blob storage account in the following regions (ZRS): <ul style="list-style-type: none"><li>• Australia East</li><li>• Central US</li><li>• East US</li><li>• East US 2</li><li>• France Central</li><li>• Japan East</li><li>• North Europe</li><li>• South Central US</li><li>• Southeast Asia</li><li>• UK South</li><li>• West Europe</li><li>• West US 2</li></ul>	60 Gbps <sup>1</sup>

RESOURCE	LIMIT
Default maximum ingress per general-purpose v2 and Blob storage account in regions that aren't listed in the previous row.	25 Gbps <sup>1</sup>
Default maximum ingress for general-purpose v1 storage accounts (all regions)	10 Gbps <sup>1</sup>
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (LRS/GRS): <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East Asia</li> <li>• East US 2</li> <li>• France Central</li> <li>• Japan East</li> <li>• Korea Central</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US</li> </ul>	120 Gbps <sup>1</sup>
Default maximum egress for general-purpose v2 and Blob storage accounts in the following regions (ZRS): <ul style="list-style-type: none"> <li>• Australia East</li> <li>• Central US</li> <li>• East US</li> <li>• East US 2</li> <li>• France Central</li> <li>• Japan East</li> <li>• North Europe</li> <li>• South Central US</li> <li>• Southeast Asia</li> <li>• UK South</li> <li>• West Europe</li> <li>• West US 2</li> </ul>	120 Gbps <sup>1</sup>
Default maximum egress for general-purpose v2 and Blob storage accounts in regions that aren't listed in the previous row.	50 Gbps <sup>1</sup>
Maximum egress for general-purpose v1 storage accounts (US regions)	20 Gbps if RA-GRS/GRS is enabled, 30 Gbps for LRS/ZRS <sup>2</sup>
Maximum egress for general-purpose v1 storage accounts (non-US regions)	10 Gbps if RA-GRS/GRS is enabled, 15 Gbps for LRS/ZRS <sup>2</sup>
Maximum number of IP address rules per storage account	200
Maximum number of virtual network rules per storage account	200

RESOURCE	LIMIT
Maximum number of resource instance rules per storage account	200
Maximum number of private endpoints per storage account	200

<sup>1</sup> Azure Storage standard accounts support higher capacity limits and higher limits for ingress and egress by request. To request an increase in account limits, contact [Azure Support](#).

<sup>2</sup> If your storage account has read-access enabled with geo-redundant storage (RA-GRS) or geo-zone-redundant storage (RA-GZRS), then the egress targets for the secondary location are identical to the egress targets of the primary location. For more information, see [Azure Storage replication](#).

For more information on limits for standard storage accounts, see [Scalability targets for standard storage accounts](#).

### Storage resource provider limits

The following limits apply only when you perform management operations by using Azure Resource Manager with Azure Storage.

RESOURCE	LIMIT
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	10 per second / 1200 per hour
Storage account management operations (list)	100 per 5 minutes

### Azure Blob storage limits

RESOURCE	TARGET
Maximum size of single blob container	Same as maximum storage account capacity
Maximum number of blocks in a block blob or append blob	50,000 blocks
Maximum size of a block in a block blob	4000 MiB
Maximum size of a block blob	50,000 X 4000 MiB (approximately 190.7 TiB)
Maximum size of a block in an append blob	4 MiB
Maximum size of an append blob	50,000 x 4 MiB (approximately 195 GiB)
Maximum size of a page blob	8 TiB <sup>2</sup>
Maximum number of stored access policies per blob container	5
Target request rate for a single blob	Up to 500 requests per second
Target throughput for a single page blob	Up to 60 MiB per second <sup>2</sup>

RESOURCE	TARGET
Target throughput for a single block blob	Up to storage account ingress/egress limits <sup>1</sup>

<sup>1</sup> Throughput for a single blob depends on several factors, including, but not limited to: concurrency, request size, performance tier, speed of source for uploads, and destination for downloads. To take advantage of the performance enhancements of [high-throughput block blobs](#), upload larger blobs or blocks. Specifically, call the [Put Blob](#) or [Put Block](#) operation with a blob or block size that is greater than 4 MiB for standard storage accounts. For premium block blob or for Data Lake Storage Gen2 storage accounts, use a block or blob size that is greater than 256 KiB.

<sup>2</sup> Page blobs are not yet supported in accounts that have the [Hierarchical namespace](#) setting on them.

The following table describes the maximum block and blob sizes permitted by service version.

SERVICE VERSION	MAXIMUM BLOCK SIZE (VIA PUT BLOCK)	MAXIMUM BLOB SIZE (VIA PUT BLOCK LIST)	MAXIMUM BLOB SIZE VIA SINGLE WRITE OPERATION (VIA PUT BLOB)
Version 2019-12-12 and later	4000 MiB	Approximately 190.7 TiB (4000 MiB X 50,000 blocks)	5000 MiB (preview)
Version 2016-05-31 through version 2019-07-07	100 MiB	Approximately 4.75 TiB (100 MiB X 50,000 blocks)	256 MiB
Versions prior to 2016-05-31	4 MiB	Approximately 195 GiB (4 MiB X 50,000 blocks)	64 MiB

## Azure Queue storage limits

RESOURCE	TARGET
Maximum size of a single queue	500 TiB
Maximum size of a message in a queue	64 KiB
Maximum number of stored access policies per queue	5
Maximum request rate per storage account	20,000 messages per second, which assumes a 1-KiB message size
Target throughput for a single queue (1-KiB messages)	Up to 2,000 messages per second

## Azure Table storage limits

The following table describes capacity, scalability, and performance targets for Table storage.

RESOURCE	TARGET
Number of tables in an Azure storage account	Limited only by the capacity of the storage account
Number of partitions in a table	Limited only by the capacity of the storage account
Number of entities in a partition	Limited only by the capacity of the storage account

RESOURCE	TARGET
Maximum size of a single table	500 TiB
Maximum size of a single entity, including all property values	1 MiB
Maximum number of properties in a table entity	255 (including the three system properties, <b>PartitionKey</b> , <b>RowKey</b> , and <b>Timestamp</b> )
Maximum total size of an individual property in an entity	Varies by property type. For more information, see <b>Property Types</b> in <a href="#">Understanding the Table Service Data Model</a> .
Size of the <b>PartitionKey</b>	A string up to 1 KiB in size
Size of the <b>RowKey</b>	A string up to 1 KiB in size
Size of an entity group transaction	A transaction can include at most 100 entities and the payload must be less than 4 MiB in size. An entity group transaction can include an update to an entity only once.
Maximum number of stored access policies per table	5
Maximum request rate per storage account	20,000 transactions per second, which assumes a 1-KiB entity size
Target throughput for a single table partition (1 KiB-entities)	Up to 2,000 entities per second

### Virtual machine disk limits

You can attach a number of data disks to an Azure virtual machine (VM). Based on the scalability and performance targets for a VM's data disks, you can determine the number and type of disk that you need to meet your performance and capacity requirements.

#### IMPORTANT

For optimal performance, limit the number of highly utilized disks attached to the virtual machine to avoid possible throttling. If all attached disks aren't highly utilized at the same time, the virtual machine can support a larger number of disks.

### For Azure managed disks:

The following table illustrates the default and maximum limits of the number of resources per region per subscription. The limits remain the same irrespective of disks encrypted with either platform-managed keys or customer-managed keys. There is no limit for the number of Managed Disks, snapshots and images per resource group.

RESOURCE	LIMIT
Standard managed disks	50,000
Standard SSD managed disks	50,000
Premium managed disks	50,000

RESOURCE	LIMIT
Standard_LRS snapshots <sup>1</sup>	75,000
Standard_ZRS snapshots <sup>1</sup>	75,000
Managed image	50,000

<sup>1</sup> The total number of full disk snapshots an individual disk may have is 200. An individual disk may also have 200 incremental snapshots, which are counted separately from full disk snapshots.

**For standard storage accounts:** A Standard storage account has a maximum total request rate of 20,000 IOPS. The total IOPS across all of your virtual machine disks in a Standard storage account should not exceed this limit.

You can roughly calculate the number of highly utilized disks supported by a single standard storage account based on the request rate limit. For example, for a Basic tier VM, the maximum number of highly utilized disks is about 66, which is 20,000/300 IOPS per disk. The maximum number of highly utilized disks for a Standard tier VM is about 40, which is 20,000/500 IOPS per disk.

**For premium storage accounts:** A premium storage account has a maximum total throughput rate of 50 Gbps. The total throughput across all of your VM disks should not exceed this limit.

For more information, see [Virtual machine sizes](#).

#### Disk encryption sets

There's a limitation of 1000 disk encryption sets per region, per subscription. For more information, see the encryption documentation for [Linux](#) or [Windows](#) virtual machines. If you need to increase the quota, contact Azure support.

### Managed virtual machine disks

#### Standard HDD managed disks

STANDARD DISK TYPE	S4	S6	S10	S15	S20	S30	S40	S50	S60	S70	S80
Disk size in GiB	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
IOPS per disk	Up to 500	Up to 1,300	Up to 2,000	Up to 2,000							
Throughput per disk	Up to 60 MB/s	Up to 300 MB/s	Up to 500 MB/s	Up to 500 MB/s							

#### Standard SSD managed disks

STA ND AR D SSD SIZ ES	E1	E2	E3	E4	E6	E10	E15	E20	E30	E40	E50	E60	E70	E80
Disk size in GiB	4	8	16	32	64	128	256	512	1,024	2,048	4,096	8,192	16,384	32,767
IOPS per disk	Up to 500	Up to 2,000	Up to 4,000	Up to 6,000										
Throughput per disk	Up to 60 MB/sec	Up to 400 MB/sec	Up to 600 MB/sec	Up to 750 MB/sec										
Max burst IOPS per disk	600	600	600	600	600	600	600	600	1000					
Max burst throughput per disk	150 MB/sec	250 MB/sec												
Max burst duration	30 min													

#### Premium SSD managed disks: Per-disk limits



PRE MIU M SSD SIZ ES	P1	P2	P3	P4	P6	P10	P15	P20	P30	P40	P50	P60	P70	P80
Eligi ble for rese rvat ion	No	No	No	No	No	No	No	No	Yes, up to one yea r	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year	Yes, up to one year

\*Applies only to disks with on-demand bursting enabled.

#### Premium SSD managed disks: Per-VM limits

RESOURCE	LIMIT
Maximum IOPS Per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/s with GS5 VM

#### Unmanaged virtual machine disks

##### Standard unmanaged virtual machine disks: Per-disk limits

VM TIER	BASIC TIER VM	STANDARD TIER VM
Disk size	4,095 GB	4,095 GB
Maximum 8-KB IOPS per persistent disk	300	500
Maximum number of disks that perform the maximum IOPS	66	40

##### Premium unmanaged virtual machine disks: Per-account limits

RESOURCE	LIMIT
Total disk capacity per account	35 TB
Total snapshot capacity per account	10 TB
Maximum bandwidth per account (ingress + egress) <sup>1</sup>	<=50 Gbps

<sup>1</sup>Ingress refers to all data from requests that are sent to a storage account. Egress refers to all data from responses that are received from a storage account.

##### Premium unmanaged virtual machine disks: Per-disk limits

PREMIUM STORAGE DISK TYPE	P10	P20	P30	P40	P50
Disk size	128 GiB	512 GiB	1,024 GiB (1 TB)	2,048 GiB (2 TB)	4,095 GiB (4 TB)

PREMIUM STORAGE DISK TYPE	P10	P20	P30	P40	P50
Maximum IOPS per disk	500	2,300	5,000	7,500	7,500
Maximum throughput per disk	100 MB/sec	150 MB/sec	200 MB/sec	250 MB/sec	250 MB/sec
Maximum number of disks per storage account	280	70	35	17	8

#### Premium unmanaged virtual machine disks: Per-VM limits

RESOURCE	LIMIT
Maximum IOPS per VM	80,000 IOPS with GS5 VM
Maximum throughput per VM	2,000 MB/sec with GS5 VM

## StorSimple System limits

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of storage account credentials	64	
Maximum number of volume containers	64	
Maximum number of volumes	255	
Maximum number of schedules per bandwidth template	168	A schedule for every hour, every day of the week.
Maximum size of a tiered volume on physical devices	64 TB for StorSimple 8100 and StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum size of a tiered volume on virtual devices in Azure	30 TB for StorSimple 8010 64 TB for StorSimple 8020	StorSimple 8010 and StorSimple 8020 are virtual devices in Azure that use Standard storage and Premium storage, respectively.
Maximum size of a locally pinned volume on physical devices	9 TB for StorSimple 8100 24 TB for StorSimple 8600	StorSimple 8100 and StorSimple 8600 are physical devices.
Maximum number of iSCSI connections	512	
Maximum number of iSCSI connections from initiators	512	

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of access control records per device	64	
Maximum number of volumes per backup policy	24	
Maximum number of backups retained per backup policy	64	
Maximum number of schedules per backup policy	10	
Maximum number of snapshots of any type that can be retained per volume	256	This amount includes local snapshots and cloud snapshots.
Maximum number of snapshots that can be present in any device	10,000	
Maximum number of volumes that can be processed in parallel for backup, restore, or clone	16	<ul style="list-style-type: none"> <li>• If there are more than 16 volumes, they're processed sequentially as processing slots become available.</li> <li>• New backups of a cloned or a restored tiered volume can't occur until the operation is finished. For a local volume, backups are allowed after the volume is online.</li> </ul>

LIMIT IDENTIFIER	LIMIT	COMMENTS
Restore and clone recover time for tiered volumes	<2 minutes	<ul style="list-style-type: none"> <li>The volume is made available within 2 minutes of a restore or clone operation, regardless of the volume size.</li> <li>The volume performance might initially be slower than normal as most of the data and metadata still resides in the cloud. Performance might increase as data flows from the cloud to the StorSimple device.</li> <li>The total time to download metadata depends on the allocated volume size. Metadata is automatically brought into the device in the background at the rate of 5 minutes per TB of allocated volume data. This rate might be affected by Internet bandwidth to the cloud.</li> <li>The restore or clone operation is complete when all the metadata is on the device.</li> <li>Backup operations can't be performed until the restore or clone operation is fully complete.</li> </ul>

LIMIT IDENTIFIER	LIMIT	COMMENTS
Restore recover time for locally pinned volumes	<2 minutes	<ul style="list-style-type: none"> <li>The volume is made available within 2 minutes of the restore operation, regardless of the volume size.</li> <li>The volume performance might initially be slower than normal as most of the data and metadata still resides in the cloud. Performance might increase as data flows from the cloud to the StorSimple device.</li> <li>The total time to download metadata depends on the allocated volume size. Metadata is automatically brought into the device in the background at the rate of 5 minutes per TB of allocated volume data. This rate might be affected by Internet bandwidth to the cloud.</li> <li>Unlike tiered volumes, if there are locally pinned volumes, the volume data is also downloaded locally on the device. The restore operation is complete when all the volume data has been brought to the device.</li> <li>The restore operations might be long and the total time to complete the restore will depend on the size of the provisioned local volume, your Internet bandwidth, and the existing data on the device. Backup operations on the locally pinned volume are allowed while the restore operation is in progress.</li> </ul>
Thin-restore availability	Last failover	
Maximum client read/write throughput, when served from the SSD tier*	920/720 MB/sec with a single 10-gigabit Ethernet network interface	Up to two times with MPIO and two network interfaces.
Maximum client read/write throughput, when served from the HDD tier*	120/250 MB/sec	
Maximum client read/write throughput, when served from the cloud tier*	11/41 MB/sec	Read throughput depends on clients generating and maintaining sufficient I/O queue depth.

\*Maximum throughput per I/O type was measured with 100 percent read and 100 percent write scenarios. Actual throughput might be lower and depends on I/O mix and network conditions.

## Stream Analytics limits

LIMIT IDENTIFIER	LIMIT	COMMENTS
Maximum number of streaming units per subscription per region	500	To request an increase in streaming units for your subscription beyond 500, contact <a href="#">Microsoft Support</a> .
Maximum number of inputs per job	60	There's a hard limit of 60 inputs per Azure Stream Analytics job.
Maximum number of outputs per job	60	There's a hard limit of 60 outputs per Stream Analytics job.
Maximum number of functions per job	60	There's a hard limit of 60 functions per Stream Analytics job.
Maximum number of streaming units per job	192	There's a hard limit of 192 streaming units per Stream Analytics job.
Maximum number of jobs per region	1,500	Each subscription can have up to 1,500 jobs per geographical region.
Reference data blob MB	5 GB	Up to 5 GB when using 6 SUs or more.
Maximum number of characters in a query	512000	There's a hard limit of 512k characters in an Azure Stream Analytics job query.

## Virtual Machines limits

### Virtual Machines limits

RESOURCE	LIMIT
Virtual machines per cloud service <sup>1</sup>	50
Input endpoints per cloud service <sup>2</sup>	150

<sup>1</sup> Virtual machines created by using the classic deployment model instead of Azure Resource Manager are automatically stored in a cloud service. You can add more virtual machines to that cloud service for load balancing and availability.

<sup>2</sup> Input endpoints allow communications to a virtual machine from outside the virtual machine's cloud service. Virtual machines in the same cloud service or virtual network can automatically communicate with each other.

### Virtual Machines limits - Azure Resource Manager

The following limits apply when you use Azure Resource Manager and Azure resource groups.

RESOURCE	LIMIT
VMs per <a href="#">subscription</a>	25,000 <sup>1</sup> per region.
VM total cores per <a href="#">subscription</a>	20 <sup>1</sup> per region. Contact support to increase limit.

RESOURCE	LIMIT
Azure Spot VM total cores per <a href="#">subscription</a>	20 <sup>1</sup> per region. Contact support to increase limit.
VM per series, such as Dv2 and F, cores per <a href="#">subscription</a>	20 <sup>1</sup> per region. Contact support to increase limit.
<a href="#">Availability sets</a> per subscription	2,500 per region.
Virtual machines per availability set	200
<a href="#">Proximity placement groups</a> per <a href="#">resource group</a>	800
Certificates per availability set	199 <sup>2</sup>
Certificates per subscription	Unlimited <sup>3</sup>

<sup>1</sup> Default limits vary by offer category type, such as Free Trial and Pay-As-You-Go, and by series, such as Dv2, F, and G. For example, the default for Enterprise Agreement subscriptions is 350. For security, subscriptions default to 20 cores to prevent large core deployments. If you need more cores, submit a support ticket.

<sup>2</sup> Properties such as SSH public keys are also pushed as certificates and count towards this limit. To bypass this limit, use the [Azure Key Vault extension for Windows](#) or the [Azure Key Vault extension for Linux](#) to install certificates.

<sup>3</sup> With Azure Resource Manager, certificates are stored in the Azure Key Vault. The number of certificates is unlimited for a subscription. There's a 1-MB limit of certificates per deployment, which consists of either a single VM or an availability set.

#### NOTE

Virtual machine cores have a regional total limit. They also have a limit for regional per-size series, such as Dv2 and F. These limits are separately enforced. For example, consider a subscription with a US East total VM core limit of 30, an A series core limit of 30, and a D series core limit of 30. This subscription can deploy 30 A1 VMs, or 30 D1 VMs, or a combination of the two not to exceed a total of 30 cores. An example of a combination is 10 A1 VMs and 20 D1 VMs.

## Shared Image Gallery limits

There are limits, per subscription, for deploying resources using Shared Image Galleries:

- 100 shared image galleries, per subscription, per region
- 1,000 image definitions, per subscription, per region
- 10,000 image versions, per subscription, per region

## Virtual machine scale sets limits

RESOURCE	LIMIT
Maximum number of VMs in a scale set	1,000
Maximum number of VMs based on a custom VM image in a scale set	600
Maximum number of scale sets in a region	2,500

## See also

- [Understand Azure limits and increases](#)
- [Virtual machine and cloud service sizes for Azure](#)
- [Sizes for Azure Cloud Services](#)
- [Naming rules and restrictions for Azure resources](#)

# Kudu service overview

11/2/2021 • 2 minutes to read • [Edit Online](#)

Kudu is the engine behind a number of features in [Azure App Service](#) related to source control based deployment, and other deployment methods like Dropbox and OneDrive sync.

## Access Kudu for your app

Anytime you create an app, App Service creates a companion app for it that's secured by HTTPS. This Kudu app is accessible at:

- App not in Isolated tier: `https://<app-name>.scm.azurewebsites.net`
- App in Isolated tier (App Service Environment): `https://<app-name>.scm.<ase-name>.p.azurewebsites.net`

For more information, see [Accessing the kudu service](#).

## Kudu features

Kudu gives you helpful information about your App Service app, such as:

- App settings
- Connection strings
- Environment variables
- Server variables
- HTTP headers

It also provides other features, such as:

- Run commands in the [Kudu console](#).
- Download IIS diagnostic dumps or Docker logs.
- Manage IIS processes and site extensions.
- Add deployment webhooks for Windows apps.
- Allow ZIP deployment UI with `/zipDeploy`.
- Generates [custom deployment scripts](#).
- Allows access with [REST API](#).

## More Resources

Kudu is an [open source project](#), and has its documentation at [Kudu Wiki](#).

# .NET migration cases for Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

Azure App Service provides easy-to-use tools to quickly discover on-premise .NET web apps, assess for readiness, and migrate both the content & supported configurations to App Service.

These tools are developed to support different kinds of scenarios, focused on discovery, assessment, and migration. Following is list of .NET migration tools and use cases.

## Migrate from multiple servers at-scale (preview)

Azure Migrate recently announced at-scale, agentless discovery, and assessment of ASP.NET web apps. You can now easily discover ASP.NET web apps running on Internet Information Services (IIS) servers in a VMware environment and assess them for migration to Azure App Service. Assessments will help you determine the web app migration readiness, migration blockers, remediation guidance, recommended SKU, and hosting costs. At-scale migration resources for are found below.

### At-scale migration resources

#### HOW-TOS

[Discover web apps and SQL Server instances](#)

[Create an Azure App Service assessment](#)

[Tutorial to assess web apps for migration to Azure App Service](#)

[Discover software inventory on on-premises servers with Azure Migrate](#)

#### Blog

[Discover and assess ASP.NET apps at-scale with Azure Migrate](#)

#### FAQ

[Azure App Service assessments in Azure Migrate Discovery and assessment tool](#)

#### Best practices

[Assessment best practices in Azure Migrate Discovery and assessment tool](#)

#### Video

[At scale discovery and assessment for ASP.NET app migration with Azure Migrate](#)

## Migrate from an IIS server

You can migrate ASP.NET web apps from single IIS server discovered through Azure Migrate's at-scale discovery experience using [PowerShell scripts \(download\)](#). Watch the video for [updates on migrating to Azure App Service](#).

# ASP.NET web app migration

Using App Service Migration Assistant, you can [migrate your standalone on-premises ASP.NET web app onto Azure App Service](#). App Service Migration Assistant is designed to simplify your journey to the cloud through a free, simple, and fast solution to migrate applications from on-premises to the cloud. For more information about the migration assistant tool, see the [FAQ](#).

## Containerize an ASP.NET web app

Some .NET Framework web applications may have dependencies to libraries and other capabilities not available in Azure App Service. These apps may rely on other components in the Global Assembly Cache. Previously, you could only run these applications on virtual machines. However, now you can run them in Azure App Service Windows Containers.

The [app containerization tool](#) can repackage applications as containers with minimal changes. The tool currently supports containerizing ASP.NET applications and Apache Tomcat Java applications. For more information about containerization and migration, see the [how-to](#).

## Next steps

[Migrate an on-premise web application to Azure App Service](#)

# Java migration resources for Azure App Service

11/2/2021 • 2 minutes to read • [Edit Online](#)

Azure App Service provides tools to discover web apps deployed to on-premise web servers. You can assess these apps for readiness, then migrate them to App Service. Both the web app content and supported configuration can be migrated to App Service. These tools are developed to support a wide variety of scenarios focused on discovery, assessment, and migration.

## Java Tomcat migration (Linux)

[Download the assistant](#) to migrate a Java app running on Apache Tomcat web server. You can also use Azure Container Registry to migrate on-premise Linux Docker containers to App Service.

### RESOURCES

#### How-tos

[Java App-Service-Migration-Assistant Wiki](#)

[Azure/App-Service-Migration-Assistant Wiki](#)

#### Videos

[Point and Migrate Java Apps to Azure App Service Using the Migration System](#)

## Standalone Tomcat Web App Migration (Windows OS)

Download this [preview tool](#) to migrate a Java web app on Apache Tomcat to App Service on Windows. For more information, see the [video](#) and [how-to](#).

## Containerize standalone Tomcat Web App

App Containerization offers a simple approach to repackage applications as containers with minimal code changes. The tool currently supports containerizing ASP.NET and Apache Tomcat Java web applications. For more information, see the [blog](#) and [how-to](#).

## Next steps

[Migrate Tomcat applications to Tomcat on Azure App Service](#)

# Best Practices for Azure App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

This article summarizes best practices for using [Azure App Service](#).

## Colocation

When Azure resources composing a solution such as a web app and a database are located in different regions, it can have the following effects:

- Increased latency in communication between resources
- Monetary charges for outbound data transfer cross-region as noted on the [Azure pricing page](#).

Colocation in the same region is best for Azure resources composing a solution such as a web app and a database or storage account used to hold content or data. When creating resources, make sure they are in the same Azure region unless you have specific business or design reason for them not to be. You can move an App Service app to the same region as your database by using the [App Service cloning feature](#) currently available for Premium App Service Plan apps.

## When apps consume more memory than expected

When you notice an app consumes more memory than expected as indicated via monitoring or service recommendations, consider the [App Service Auto-Healing feature](#). One of the options for the Auto-Healing feature is taking custom actions based on a memory threshold. Actions span the spectrum from email notifications to investigation via memory dump to on-the-spot mitigation by recycling the worker process. Auto-healing can be configured via web.config and via a friendly user interface as described at in this blog post for the [App Service Support Site Extension](#).

## When apps consume more CPU than expected

When you notice an app consumes more CPU than expected or experiences repeated CPU spikes as indicated via monitoring or service recommendations, consider scaling up or scaling out the App Service plan. If your application is stateful, scaling up is the only option, while if your application is stateless, scaling out gives you more flexibility and higher scale potential.

For more information about "stateful" vs "stateless" applications you can watch this video: [Planning a Scalable End-to-End Multi-Tier Application on Azure App Service](#). For more information about App Service scaling and autoscaling options, see [Scale a Web App in Azure App Service](#).

## When socket resources are exhausted

A common reason for exhausting outbound TCP connections is the use of client libraries, which are not implemented to reuse TCP connections, or when a higher-level protocol such as HTTP - Keep-Alive is not used. Review the documentation for each of the libraries referenced by the apps in your App Service Plan to ensure they are configured or accessed in your code for efficient reuse of outbound connections. Also follow the library documentation guidance for proper creation and release or cleanup to avoid leaking connections. While such client libraries investigations are in progress, impact may be mitigated by scaling out to multiple instances.

### Node.js and outgoing http requests

When working with Node.js and many outgoing http requests, dealing with HTTP - Keep-Alive is important. You can use the [agentkeepalive](#) `npm` package to make it easier in your code.

Always handle the `http` response, even if you do nothing in the handler. If you don't handle the response properly, your application gets stuck eventually because no more sockets are available.

For example, when working with the `http` or `https` package:

```
const request = https.request(options, function(response) {
 response.on('data', function() { /* do nothing */ });
});
```

If you are running on App Service on Linux on a machine with multiple cores, another best practice is to use PM2 to start multiple Node.js processes to execute your application. You can do it by specifying a startup command to your container.

For example, to start four instances:

```
pm2 start /home/site/wwwroot/app.js --no-daemon -i 4
```

## When your app backup starts failing

The two most common reasons why app backup fails are: invalid storage settings and invalid database configuration. These failures typically happen when there are changes to storage or database resources, or changes for how to access these resources (for example, credentials updated for the database selected in the backup settings). Backups typically run on a schedule and require access to storage (for outputting the backed-up files) and databases (for copying and reading contents to be included in the backup). The result of failing to access either of these resources would be consistent backup failure.

When backup failures happen, review most recent results to understand which type of failure is happening. For storage access failures, review and update the storage settings used in the backup configuration. For database access failures, review and update your connection strings as part of app settings; then proceed to update your backup configuration to properly include the required databases. For more information on app backups, see [Back up a web app in Azure App Service](#).

## When new Node.js apps are deployed to Azure App Service

Azure App Service default configuration for Node.js apps is intended to best suit the needs of most common apps. If configuration for your Node.js app would benefit from personalized tuning to improve performance or optimize resource usage for CPU/memory/network resources, see [Best practices and troubleshooting guide for Node applications on Azure App Service](#). This article describes the iisnode settings you may need to configure for your Node.js app, describes the various scenarios or issues that your app may be facing, and shows how to address these issues.

### Next Steps

For more information on best practices, visit [App Service Diagnostics](#) to find out actionable best practices specific to your resource.

- Navigate to your Web App in the [Azure portal](#).
- Click on **Diagnose and solve problems** in the left navigation, which opens App Service Diagnostics.
- Choose **Best Practices** homepage tile.
- Click **Best Practices for Availability & Performance** or **Best Practices for Optimal Configuration** to view the current state of your app in regards to these best practices.

You can also use this link to directly open App Service Diagnostics for your resource:

```
https://ms.portal.azure.com/?websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourc
```

# Troubleshooting intermittent outbound connection errors in Azure App Service

11/2/2021 • 8 minutes to read • [Edit Online](#)

This article helps you troubleshoot intermittent connection errors and related performance issues in [Azure App Service](#). This topic will provide more information on, and troubleshooting methodologies for, exhaustion of source address network translation (SNAT) ports. If you require more help at any point in this article, contact the Azure experts at the [MSDN Azure and the Stack Overflow forums](#). Alternatively, file an Azure support incident. Go to the [Azure Support site](#) and select **Get Support**.

## Symptoms

Applications and Functions hosted on Azure App service may exhibit one or more of the following symptoms:

- Slow response times on all or some of the instances in a service plan.
- Intermittent 5xx or **Bad Gateway** errors
- Timeout error messages
- Could not connect to external endpoints (like SQLDB, Service Fabric, other App services etc.)

## Cause

The major cause for intermittent connection issues is hitting a limit while making new outbound connections.

The limits you can hit include:

- TCP Connections: There is a limit on the number of outbound connections that can be made. The limit on outbound connections is associated with the size of the worker used.
- SNAT ports: [Outbound connections in Azure](#) describes SNAT port restrictions and how they affect outbound connections. Azure uses source network address translation (SNAT) and Load Balancers (not exposed to customers) to communicate with public IP addresses. Each instance on Azure App service is initially given a pre-allocated number of 128 SNAT ports. The SNAT port limit affects opening connections to the same address and port combination. If your app creates connections to a mix of address and port combinations, you will not use up your SNAT ports. The SNAT ports are used up when you have repeated calls to the same address and port combination. Once a port has been released, the port is available for reuse as needed. The Azure Network load balancer reclaims SNAT port from closed connections only after waiting for 4 minutes.

When applications or functions rapidly open a new connection, they can quickly exhaust their pre-allocated quota of the 128 ports. They are then blocked until a new SNAT port becomes available, either through dynamically allocating additional SNAT ports, or through reuse of a reclaimed SNAT port. If your app runs out of SNAT ports, it will have intermittent outbound connectivity issues.

## Avoiding the problem

There are a few solutions that let you avoid SNAT port limitations. They include:

- connection pools: By pooling your connections, you avoid opening new network connections for calls to the same address and port.
- service endpoints: You don't have a SNAT port restriction to the services secured with service endpoints.
- private endpoints: You don't have a SNAT port restriction to services secured with private endpoints.
- NAT gateway: With a NAT gateway, you have 64k outbound SNAT ports that are usable by the resources

sending traffic through it.

Avoiding the SNAT port problem means avoiding the creation of new connections repetitively to the same host and port. Connection pools are one of the more obvious ways to solve that problem.

If your destination is an Azure service that supports service endpoints, you can avoid SNAT port exhaustion issues by using [regional VNet Integration](#) and service endpoints or private endpoints. When you use regional VNet Integration and place service endpoints on the integration subnet, your app outbound traffic to those services will not have outbound SNAT port restrictions. Likewise, if you use regional VNet Integration and private endpoints, you will not have any outbound SNAT port issues to that destination.

If your destination is an external endpoint outside of Azure, [using a NAT gateway](#) gives you 64k outbound SNAT ports. It also gives you a dedicated outbound address that you don't share with anybody.

If possible, improve your code to use connection pools and avoid the entire situation. It isn't always possible to change code fast enough to mitigate this situation. For the cases where you can't change your code in time, take advantage of the other solutions. The best solution to the problem is to combine all of the solutions as best you can. Try to use service endpoints and private endpoints to Azure services and the NAT gateway for the rest.

General strategies for mitigating SNAT port exhaustion are discussed in the [Problem-solving section](#) of the [Outbound connections of Azure](#) documentation. Of these strategies, the following are applicable to apps and functions hosted on Azure App service.

### Modify the application to use connection pooling

- For pooling HTTP connections, review [Pool HTTP connections with HttpClientFactory](#).
- For information on SQL Server connection pooling, review [SQL Server Connection Pooling \(ADO.NET\)](#).
- For implementing pooling with entity framework applications, review [DbContext pooling](#).

Here is a collection of links for implementing Connection pooling by different solution stack.

#### Node

By default, connections for NodeJS are not kept alive. Below are the popular databases and packages for connection pooling which contain examples for how to implement them.

- [MySQL](#)
- [MongoDB](#)
- [PostgreSQL](#)
- [SQL Server](#)

#### HTTP Keep-alive

- [agentkeepalive](#)
- [Node.js v13.9.0 Documentation](#)

#### Java

Below are the popular libraries used for JDBC connection pooling which contain examples for how to implement them: JDBC Connection Pooling.

- [Tomcat 8](#)
- [C3p0](#)
- [HikariCP](#)
- [Apache DBCP](#)

#### HTTP Connection Pooling

- [Apache Connection Management](#)
- [Class PoolingHttpClientConnectionManager](#)

## PHP

Although PHP does not support connection pooling, you can try using persistent database connections to your back-end server.

- MySQL server
  - [MySQLi connections](#) for newer versions
  - [mysql\\_pconnect](#) for older versions of PHP
- Other data Sources
  - [PHP Connection Management](#)

### Modify the application to reuse connections

- For additional pointers and examples on managing connections in Azure functions, review [Manage connections in Azure Functions](#).

### Modify the application to use less aggressive retry logic

- For additional guidance and examples, review [Retry pattern](#).

### Use keepalives to reset the outbound idle timeout

- For implementing keepalives for Node.js apps, review [My node application is making excessive outbound calls](#).

### Additional guidance specific to App Service:

- A [load test](#) should simulate real world data in a steady feeding speed. Testing apps and functions under real world stress can identify and resolve SNAT port exhaustion issues ahead of time.
- Ensure that the back-end services can return responses quickly. For troubleshooting performance issues with Azure SQL Database, review [Troubleshoot Azure SQL Database performance issues with Intelligent Insights](#).
- Scale out the App Service plan to more instances. For more information on scaling, see [Scale an app in Azure App Service](#). Each worker instance in an app service plan is allocated a number of SNAT ports. If you spread your usage across more instances, you might get the SNAT port usage per instance below the recommended limit of 100 outbound connections, per unique remote endpoint.
- Consider moving to [App Service Environment \(ASE\)](#), where you are allotted a single outbound IP address, and the limits for connections and SNAT ports are much higher. In an ASE, the number of SNAT ports per instance is based on the [Azure load balancer preallocation table](#) - so for example an ASE with 1-50 worker instances has 1024 preallocated ports per instance, while an ASE with 51-100 worker instances has 512 preallocated ports per instance.

Avoiding the outbound TCP limits is easier to solve, as the limits are set by the size of your worker. You can see the limits in [Sandbox Cross VM Numerical Limits - TCP Connections](#)

LIMIT NAME	DESCRIPTION	SMALL (A1)	MEDIUM (A2)	LARGE (A3)	ISOLATED TIER (ASE)
Connections	Number of connections across entire VM	1920	3968	8064	16,000

To avoid outbound TCP limits, you can either increase the size of your workers, or scale out horizontally.

## Troubleshooting

Knowing the two types of outbound connection limits, and what your app does, should make it easier to troubleshoot. If you know that your app makes many calls to the same storage account, you might suspect a SNAT limit. If your app creates a great many calls to endpoints all over the internet, you would suspect you are

reaching the VM limit.

If you do not know the application behavior enough to determine the cause quickly, there are some tools and techniques available in App Service to help with that determination.

### Find SNAT port allocation information

You can use [App Service Diagnostics](#) to find SNAT port allocation information, and observe the SNAT ports allocation metric of an App Service site. To find SNAT port allocation information, follow the following steps:

1. To access App Service diagnostics, navigate to your App Service web app or App Service Environment in the [Azure portal](#). In the left navigation, select **Diagnose and solve problems**.
2. Select Availability and Performance Category
3. Select SNAT Port Exhaustion tile in the list of available tiles under the category. The practice is to keep it below 128. If you do need it, you can still open a support ticket and the support engineer will get the metric from back-end for you.

Since SNAT port usage is not available as a metric, it is not possible to either autoscale based on SNAT port usage, or to configure auto scale based on SNAT ports allocation metric.

### TCP Connections and SNAT Ports

TCP connections and SNAT ports are not directly related. A TCP connections usage detector is included in the Diagnose and Solve Problems blade of any App Service site. Search for the phrase "TCP connections" to find it.

- The SNAT Ports are only used for external network flows, while the total TCP Connections includes local loopback connections.
- A SNAT port can be shared by different flows, if the flows are different in either protocol, IP address or port. The TCP Connections metric counts every TCP connection.
- The TCP connections limit happens at the worker instance level. The Azure Network outbound load balancing doesn't use the TCP Connections metric for SNAT port limiting.
- The TCP connections limits are described in [Sandbox Cross VM Numerical Limits - TCP Connections](#)
- Existing TCP sessions will fail when new outbound TCP sessions from Azure App Service source port. You can either use a single IP or reconfigure backend pool members to avoid conflicts

LIMIT NAME	DESCRIPTION	SMALL (A1)	MEDIUM (A2)	LARGE (A3)	ISOLATED TIER (ASE)
Connections	Number of connections across entire VM	1920	3968	8064	16,000

### WebJobs and Database connections

If SNAT ports are exhausted, where WebJobs are unable to connect to SQL Database, there is no metric to show how many connections are opened by each individual web application process. To find the problematic WebJob, move several WebJobs out to another App Service plan to see if the situation improves, or if an issue remains in one of the plans. Repeat the process until you find the problematic WebJob.

## Additional information

- [SNAT with App Service](#)
- [Troubleshoot slow app performance issues in Azure App Service](#)

# Troubleshoot an app in Azure App Service using Visual Studio

11/2/2021 • 25 minutes to read • [Edit Online](#)

## Overview

This tutorial shows how to use Visual Studio tools to help debug an app in [App Service](#), by running in [debug mode](#) remotely or by viewing application logs and web server logs.

You'll learn:

- Which app management functions are available in Visual Studio.
- How to use Visual Studio remote view to make quick changes in a remote app.
- How to run debug mode remotely while a project is running in Azure, both for an app and for a WebJob.
- How to create application trace logs and view them while the application is creating them.
- How to view web server logs, including detailed error messages and failed request tracing.
- How to send diagnostic logs to an Azure Storage account and view them there.

If you have Visual Studio Ultimate, you can also use [IntelliTrace](#) for debugging. IntelliTrace is not covered in this tutorial.

## Prerequisites

This tutorial works with the development environment, web project, and App Service app that you set up in [Create an ASP.NET app in Azure App Service](#). For the WebJobs sections, you'll need the application that you create in [Get Started with the Azure WebJobs SDK](#).

The code samples shown in this tutorial are for a C# MVC web application, but the troubleshooting procedures are the same for Visual Basic and Web Forms applications.

The tutorial assumes you're using Visual Studio 2019.

The streaming logs feature only works for applications that target .NET Framework 4 or later.

## App configuration and management

Visual Studio provides access to a subset of the app management functions and configuration settings available in the [Azure portal](#). In this section, you'll see what's available by using **Server Explorer**. To see the latest Azure integration features, try out **Cloud Explorer** also. You can open both windows from the **View** menu.

1. If you aren't already signed in to Azure in Visual Studio, right-click **Azure** and select **Connect to Microsoft Azure Subscription** in **Server Explorer**.

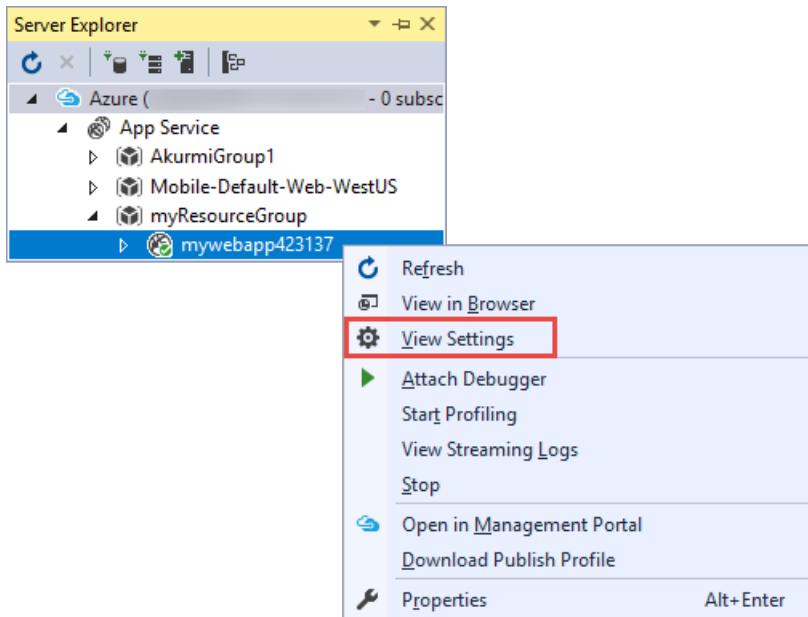
An alternative is to install a management certificate that enables access to your account. If you choose to install a certificate, right-click the **Azure** node in **Server Explorer**, and then select **Manage and Filter Subscriptions** in the context menu. In the **Manage Microsoft Azure Subscriptions** dialog box, click the **Certificates** tab, and then click **Import**. Follow the directions to download and then import a subscription file (also called a *.publishsettings* file) for your Azure account.

**NOTE**

If you download a subscription file, save it to a folder outside your source code directories (for example, in the Downloads folder), and then delete it once the import has completed. A malicious user who gains access to the subscription file can edit, create, and delete your Azure services.

For more information about connecting to Azure resources from Visual Studio, see [Assign Azure roles using the Azure portal](#).

2. In **Server Explorer**, expand **Azure** and expand **App Service**.
3. Expand the resource group that includes the app that you created in [Create an ASP.NET app in Azure App Service](#), and then right-click the app node and click **View Settings**.



The **Azure Web App** tab appears, and you can see there the app management and configuration tasks that are available in Visual Studio.

mywebapp423137: Azure Web App

Configuration Save Refresh

Actions

- Open in Management Portal
- Stop Web App
- Restart Web App

Web App Settings [Learn more](#)

.NET Framework Version	v4.5
Web Server Logging	Off
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Off
Remote Debugging	Off

Connection Strings

Name	Connection String	Database Type

Add Remove

Application Settings

Name	Value
WEBSITE_NODE_DEFAULT_VERSION	6.9.1

Add Remove

In this tutorial, you'll use the logging and tracing drop-downs. You'll also use remote debugging but you'll use a different method to enable it.

For information about the App Settings and Connection Strings boxes in this window, see [Azure App Service: How Application Strings and Connection Strings Work](#).

If you want to perform an app management task that can't be done in this window, click **Open in Management Portal** to open a browser window to the Azure portal.

## Access app files in Server Explorer

You typically deploy a web project with the `customErrors` flag in the Web.config file set to `On` or `RemoteOnly`, which means you don't get a helpful error message when something goes wrong. For many errors, all you get is a page like one of the following ones:

**Server Error in '/' Application:**

The screenshot shows a browser window with the URL <http://myexample6443.azurewe...> in the address bar. The title bar says "Runtime Error". The main content area displays the following text:

**Server Error in '/' Application.**

**Runtime Error**

**Description:** An application error occurred on the server. The current custom error settings for this application prevent the details of the application error from being viewed remotely (for security reasons). It could, however, be viewed by browsers running on the local server machine.

**Details:** To enable the details of this specific error message to be viewable on remote machines, please create a <customErrors> tag within a "web.config" configuration file located in the root directory of the current web application. This <customErrors> tag should then have its "mode" attribute set to "Off".

```
<!-- Web.Config Configuration File -->

<configuration>
 <system.web>
 <customErrors mode="Off"/>
 </system.web>
</configuration>
```

**Notes:** The current error page you are seeing can be replaced by a custom error page by modifying the "defaultRedirect" attribute of the application's <customErrors> configuration tag to point to a custom error page URL.

```
<!-- Web.Config Configuration File -->

<configuration>
 <system.web>
 <customErrors mode="RemoteOnly" defaultRedirect="mycustompage.htm"/>
 </system.web>
</configuration>
```

An error occurred:

The screenshot shows a browser window with the URL <http://myexample6443.azurewe...> in the address bar. The title bar says "Error - My ASP.NET Applica...". The main content area displays the following text:

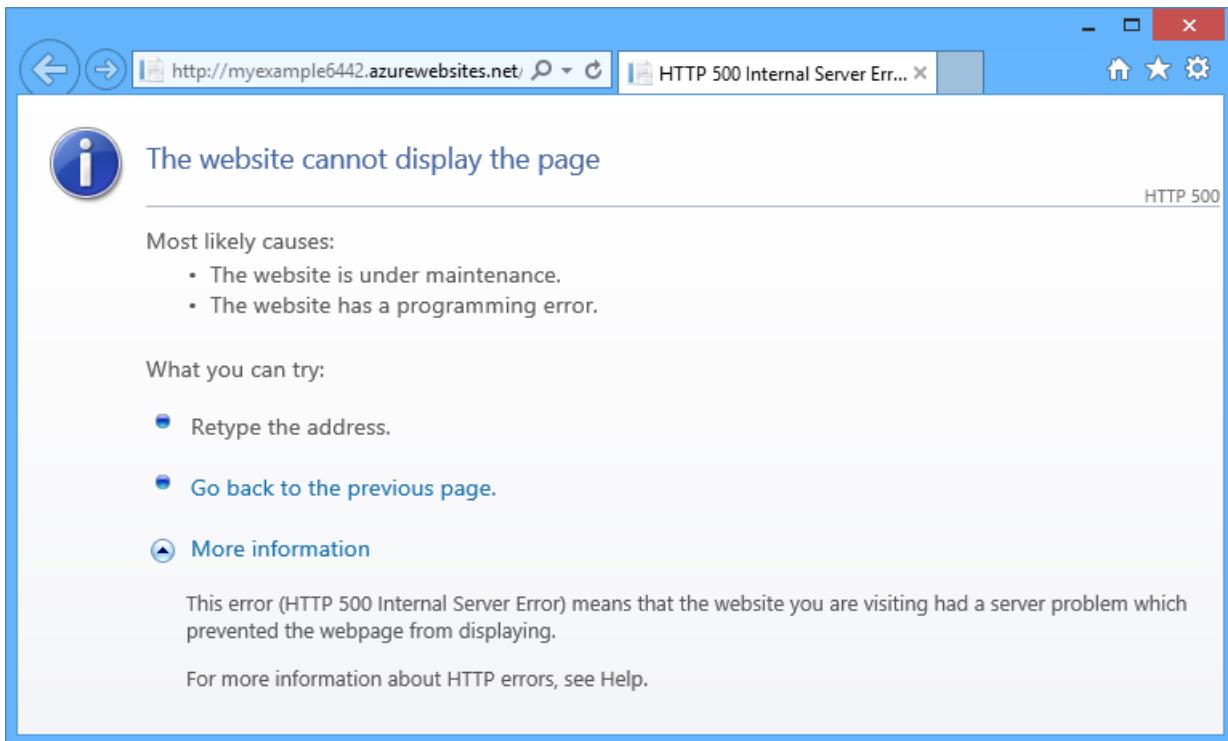
Application name

**Error.**

**An error occurred while processing your request.**

© 2014 - My ASP.NET Application

The website cannot display the page

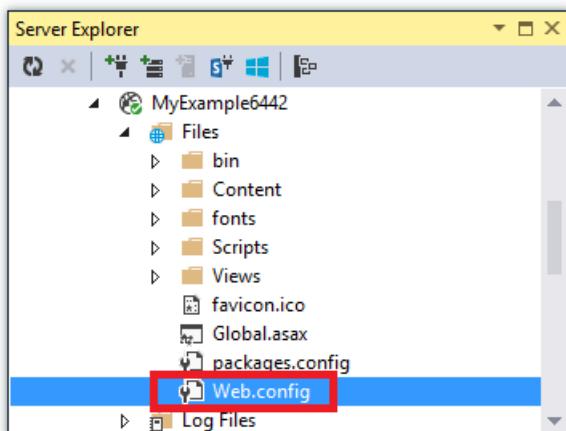


Frequently the easiest way to find the cause of the error is to enable detailed error messages, which the first of the preceding screenshots explains how to do. That requires a change in the deployed Web.config file. You could edit the *Web.config* file in the project and redeploy the project, or create a [Web.config transform](#) and deploy a debug build, but there's a quicker way: in **Solution Explorer**, you can directly view and edit files in the remote app by using the *remote view* feature.

1. In **Server Explorer**, expand **Azure**, expand **App Service**, expand the resource group that your app is located in, and then expand the node for your app.

You see nodes that give you access to the app's content files and log files.

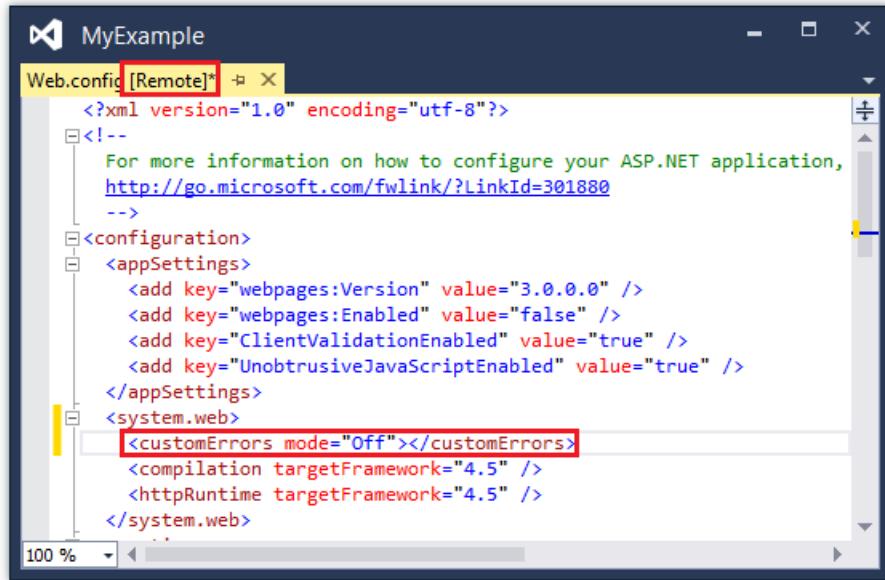
2. Expand the **Files** node, and double-click the *Web.config* file.



Visual Studio opens the *Web.config* file from the remote app and shows [Remote] next to the file name in the title bar.

3. Add the following line to the `system.web` element:

```
<customErrors mode="Off"></customErrors>
```



4. Refresh the browser that is showing the unhelpful error message, and now you get a detailed error message, such as the following example:

The screenshot shows a browser window displaying a server error. The title bar says 'http://myexample6442.azurewebsite...'. The main content is:

## Server Error in '/' Application.

**Cannot convert null to 'int' because it is a non-nullable value type**

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** Microsoft.CSharp.RuntimeBinder.RuntimeBinderException: Cannot convert null to 'int' because it is a non-nullable value type

**Source Error:**

```
Line 1: @{
Line 2: ViewBag.Title = "Home Page";
Line 3: int x = ViewBag.Error;
Line 4: }
Line 5:
```

**Source File:** d:\home\site\wwwroot\Views\Home\Index.cshtml **Line:** 3

**Stack Trace:**

```
[RuntimeBinderException: Cannot convert null to 'int' because it is a non-nullable v
CallSite.Target(Closure , CallSite , Object) +115
```

(The error shown was created by adding the line shown in red to *Views\Home\Index.cshtml*.)

Editing the Web.config file is only one example of scenarios in which the ability to read and edit files on your App Service app make troubleshooting easier.

## Remote debugging apps

If the detailed error message doesn't provide enough information, and you can't re-create the error locally, another way to troubleshoot is to run in debug mode remotely. You can set breakpoints, manipulate memory directly, step through code, and even change the code path.

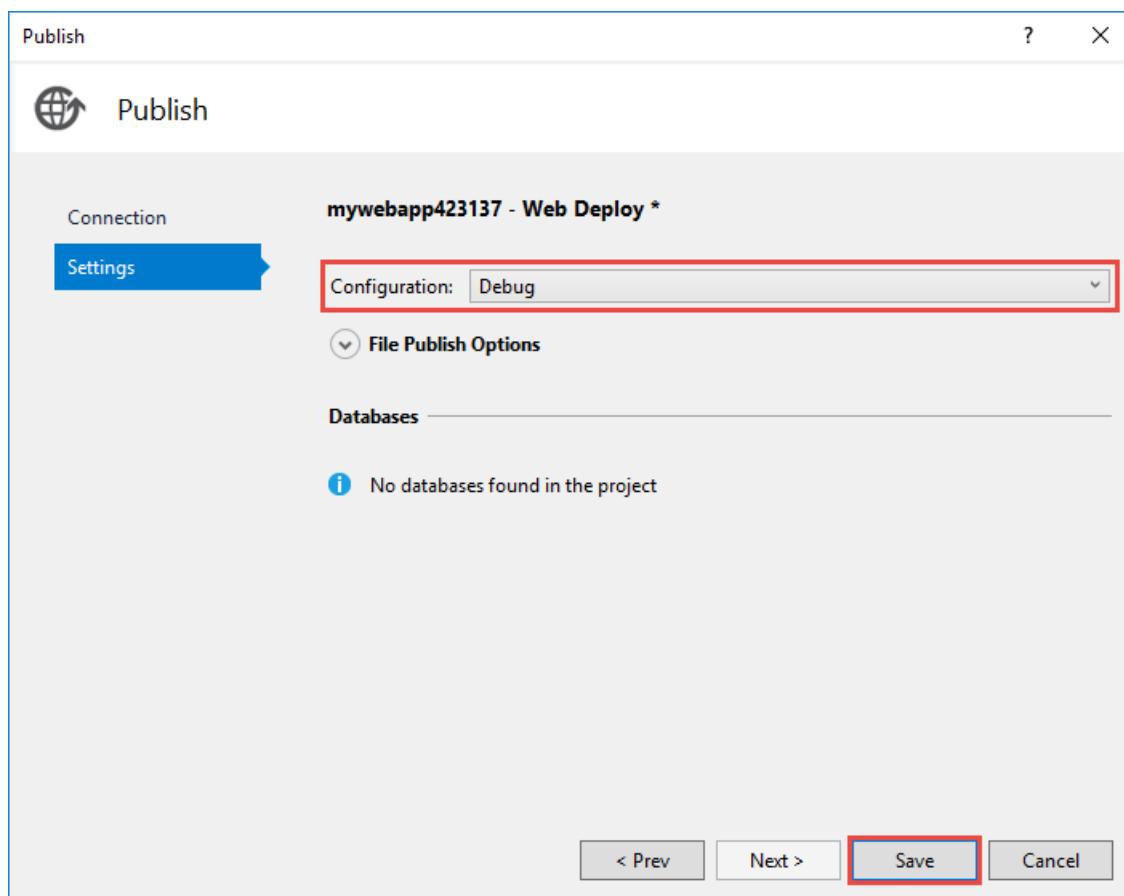
Remote debugging does not work in Express editions of Visual Studio.

This section shows how to debug remotely using the project you create in [Create an ASP.NET app in Azure App Service](#).

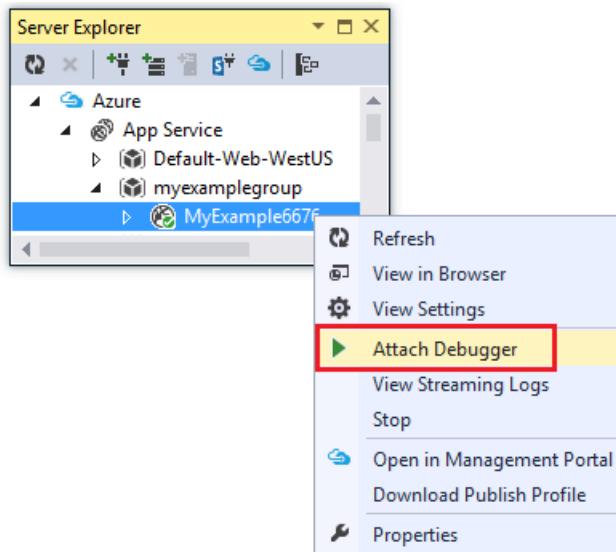
1. Open the web project that you created in [Create an ASP.NET app in Azure App Service](#).
2. Open *Controllers\HomeController.cs*.
3. Delete the `About()` method and insert the following code in its place.

```
public ActionResult About()
{
 string currentTime = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss");
 ViewBag.Message = "The current time is " + currentTime;
 return View();
}
```

4. Set a breakpoint on the `ViewBag.Message` line.
5. In **Solution Explorer**, right-click the project, and click **Publish**.
6. In the **Profile** drop-down list, select the same profile that you used in [Create an ASP.NET app in Azure App Service](#). Then, click **Settings**.
7. In the **Publish** dialog, click the **Settings** tab, and then change **Configuration** to **Debug**, and then click **Save**.



8. Click **Publish**. After deployment finishes and your browser opens to the Azure URL of your app, close the browser.
9. In **Server Explorer**, right-click your app, and then click **Attach Debugger**.



The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on an app in a 48-hour period. When you start debugging again in the same period, there isn't a delay.

**NOTE**

If you have any trouble starting the debugger, try to do it by using **Cloud Explorer** instead of **Server Explorer**.

10. Click **About** in the menu.

Visual Studio stops on the breakpoint, and the code is running in Azure, not on your local computer.

11. Hover over the `currentTime` variable to see the time value.

```
0 references
public ActionResult About()
{
 string currentTime = DateTime.Now.ToString();
 ViewBag.Message = "The current time is " + currentTime;

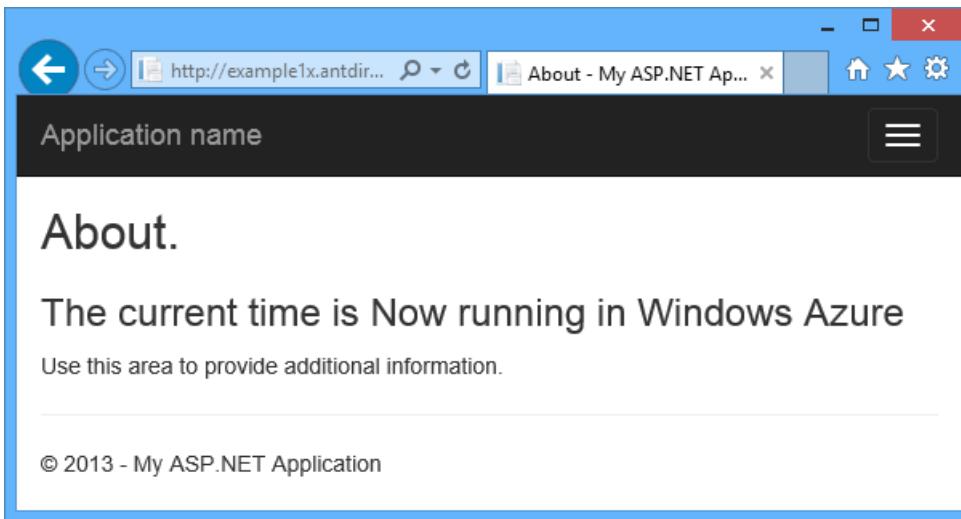
 return View();
}
```

The time you see is the Azure server time, which may be in a different time zone than your local computer.

12. Enter a new value for the `currentTime` variable, such as "Now running in Azure".

13. Press F5 to continue running.

The About page running in Azure displays the new value that you entered into the `currentTime` variable.



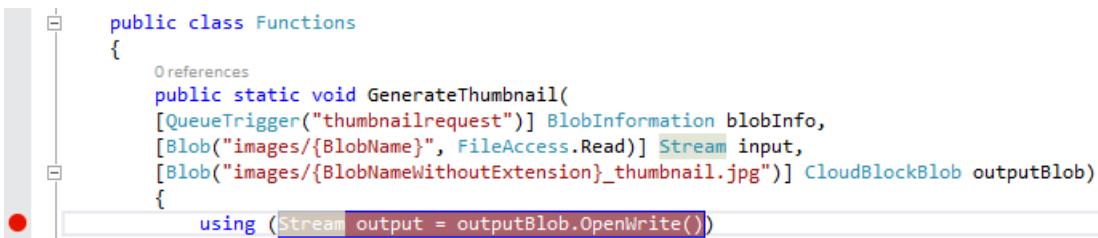
## Remote debugging WebJobs

This section shows how to debug remotely using the project and app you create in [Get Started with the Azure WebJobs SDK](#).

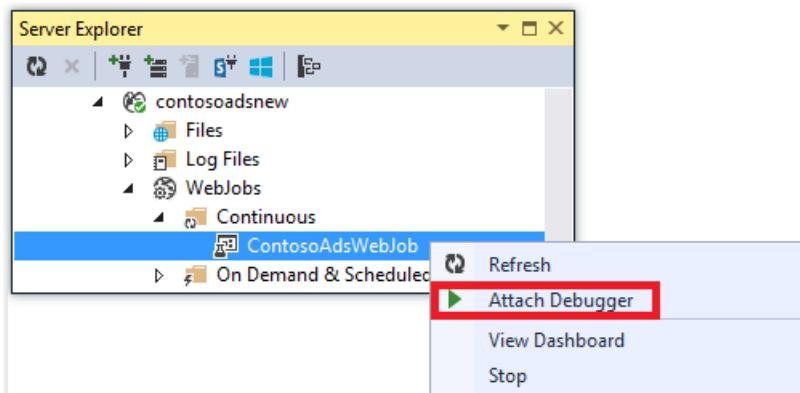
The features shown in this section are available only in Visual Studio 2013 with Update 4 or later.

Remote debugging only works with continuous WebJobs. Scheduled and on-demand WebJobs don't support debugging.

1. Open the web project that you created in [Get Started with the Azure WebJobs SDK](#).
2. In the ContosoAdsWebJob project, open *Functions.cs*.
3. Set a breakpoint on the first statement in the `GnerateThumbnail` method.



4. In **Solution Explorer**, right-click the web project (not the WebJob project), and click **Publish**.
  5. In the **Profile** drop-down list, select the same profile that you used in [Get Started with the Azure WebJobs SDK](#).
  6. Click the **Settings** tab, and change **Configuration** to **Debug**, and then click **Publish**.
- Visual Studio deploys the web and WebJob projects, and your browser opens to the Azure URL of your app.
7. In **Server Explorer**, expand **Azure > App Service > your resource group > your app > WebJobs > Continuous**, and then right-click **ContosoAdsWebJob**.
  8. Click **Attach Debugger**.

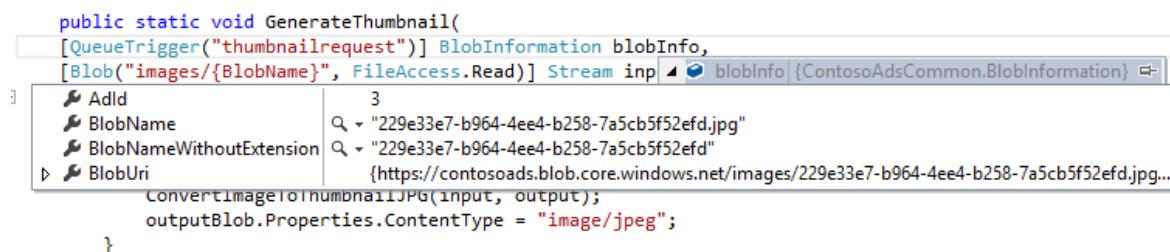


The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on an app in a 48-hour period. When you start debugging again in the same period, there isn't a delay.

9. In the web browser that is opened to the Contoso Ads home page, create a new ad.

Creating an ad causes a queue message to be created, which is picked up by the WebJob and processed. When the WebJobs SDK calls the function to process the queue message, the code hits your breakpoint.

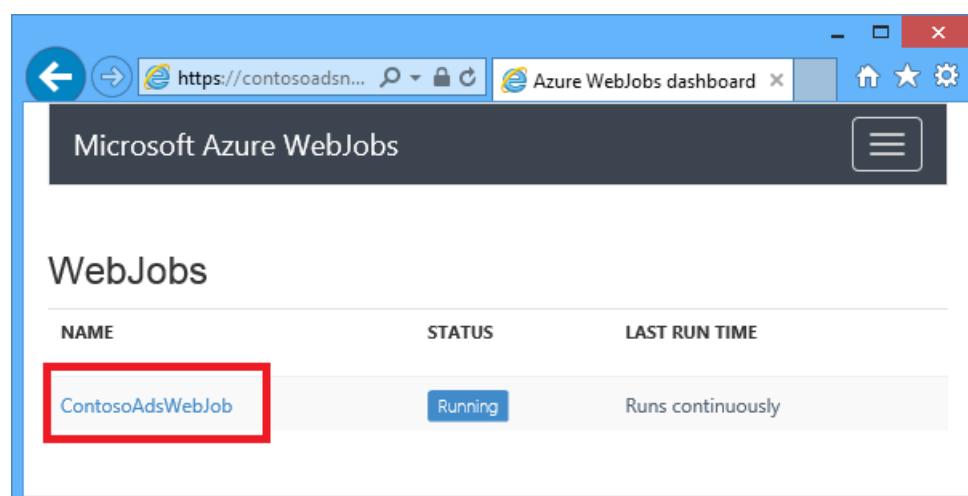
10. When the debugger breaks at your breakpoint, you can examine and change variable values while the program is running in the cloud. In the following illustration, the debugger shows the contents of the `blobInfo` object that was passed to the `GenerateThumbnail` method.



11. Press F5 to continue running.

The `GenerateThumbnail` method finishes creating the thumbnail.

12. In the browser, refresh the Index page and you see the thumbnail.
13. In Visual Studio, press SHIFT+F5 to stop debugging.
14. In **Server Explorer**, right-click the ContosoAdsWebJob node and click **View Dashboard**.
15. Sign in with your Azure credentials, and then click the WebJob name to go to the page for your WebJob.



The Dashboard shows that the `GenerateThumbnail` function executed recently.

(The next time you click **View Dashboard**, you don't have to sign in, and the browser goes directly to the page for your WebJob.)

16. Click the function name to see details about the function execution.

The screenshot shows the Azure WebJobs dashboard with the URL `https://contosoadsns... Azure WebJobs dashboard`. The main content area displays the **Invocation Details** for the function `Functions.GenerateThumbnail`. A green box indicates a **Success** status 9 minutes ago (54 seconds running time). Below this, a note says a new queue message was detected on 'thumbnailrequest'. The table shows parameters and their values:

PARAMETER	VALUE	NOTES
<code>blobInfo</code>	<code>{"BlobUri": "https://contosoads.blob.core.windows.net/images/1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5.jpg", "BlobName": "1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5.jpg", "BlobNameWithoutExtension": "1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5", "AdId": 5}</code>	Read 116,652 bytes (100.15% of total). (about 596 milliseconds spent on I/O)
<code>input</code>	<code>images/1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5.jpg</code>	
<code>outputBlob</code>	<code>images/1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5_thumbnail.jpg</code>	

A **Toggle Output** button is at the bottom left.

If your function **wrote logs**, you could click **ToggleOutput** to see them.

## Notes about remote debugging

- Running in debug mode in production is not recommended. If your production app is not scaled out to multiple server instances, debugging prevents the web server from responding to other requests. If you do have multiple web server instances, when you attach to the debugger, you get a random instance, and you have no way to ensure that subsequent browser requests go to the same instance. Also, you typically don't deploy a debug build to production, and compiler optimizations for release builds might make it impossible to show what is happening line by line in your source code. For troubleshooting production problems, your best resource is application tracing and web server logs.
- Avoid long stops at breakpoints when remote debugging. Azure treats a process that is stopped for longer than a few minutes as an unresponsive process, and shuts it down.

- While you're debugging, the server is sending data to Visual Studio, which could affect bandwidth charges. For information about bandwidth rates, see [Azure Pricing](#).
- Make sure that the `debug` attribute of the `compilation` element in the `Web.config` file is set to true. It is set to true by default when you publish a debug build configuration.

```
<system.web>
 <compilation debug="true" targetFramework="4.5" />
 <httpRuntime targetFramework="4.5" />
</system.web>
```

- If you find that the debugger doesn't step into the code that you want to debug, you might have to change the Just My Code setting. For more information, see [Specify whether to debug only user code using Just My Code in Visual Studio](#).
- A timer starts on the server when you enable the remote debugging feature, and after 48 hours the feature is automatically turned off. This 48-hour limit is done for security and performance reasons. You can easily turn the feature back on as many times as you like. We recommend leaving it disabled when you are not actively debugging.
- You can manually attach the debugger to any process, not only the app process (w3wp.exe). For more information about how to use debug mode in Visual Studio, see [Debugging in Visual Studio](#).

## Diagnostic logs overview

An ASP.NET application that runs in an App Service app can create the following kinds of logs:

- Application tracing logs**  
The application creates these logs by calling methods of the `System.Diagnostics.Trace` class.
- Web server logs**  
The web server creates a log entry for every HTTP request to the app.
- Detailed error message logs**  
The web server creates an HTML page with some additional information for failed HTTP requests (requests that result in status code 400 or greater).
- Failed request tracing logs**  
The web server creates an XML file with detailed tracing information for failed HTTP requests. The web server also provides an XSL file to format the XML in a browser.

Logging affects app performance, so Azure gives you the ability to enable or disable each type of log as needed. For application logs, you can specify that only logs above a certain severity level should be written. When you create a new app, by default all logging is disabled.

Logs are written to files in a `LogFiles` folder in the file system of your app and are accessible via FTP. Web server logs and application logs can also be written to an Azure Storage account. You can retain a greater volume of logs in a storage account than is possible in the file system. You're limited to a maximum of 100 megabytes of logs when you use the file system. (File system logs are only for short-term retention. Azure deletes old log files to make room for new ones after the limit is reached.)

## Create and view application trace logs

In this section, you do the following tasks:

- Add tracing statements to the web project that you created in [Get started with Azure and ASP.NET](#).
- View the logs when you run the project locally.
- View the logs as they are generated by the application running in Azure.

For information about how to create application logs in WebJobs, see [How to work with Azure queue storage using the WebJobs SDK - How to write logs](#). The following instructions for viewing logs and controlling how they're stored in Azure apply also to application logs created by WebJobs.

## Add tracing statements to the application

1. Open `Controllers\HomeController.cs`, and replace the `Index`, `About`, and `Contact` methods with the following code in order to add `Trace` statements and a `using` statement for `System.Diagnostics`:

```
public ActionResult Index()
{
 Trace.WriteLine("Entering Index method");
 ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";
 Trace.TraceInformation("Displaying the Index page at " + DateTime.Now.ToString());
 Trace.WriteLine("Leaving Index method");
 return View();
}

public ActionResult About()
{
 Trace.WriteLine("Entering About method");
 ViewBag.Message = "Your app description page.";
 Trace.TraceWarning("Transient error on the About page at " + DateTime.Now.ToString());
 Trace.WriteLine("Leaving About method");
 return View();
}

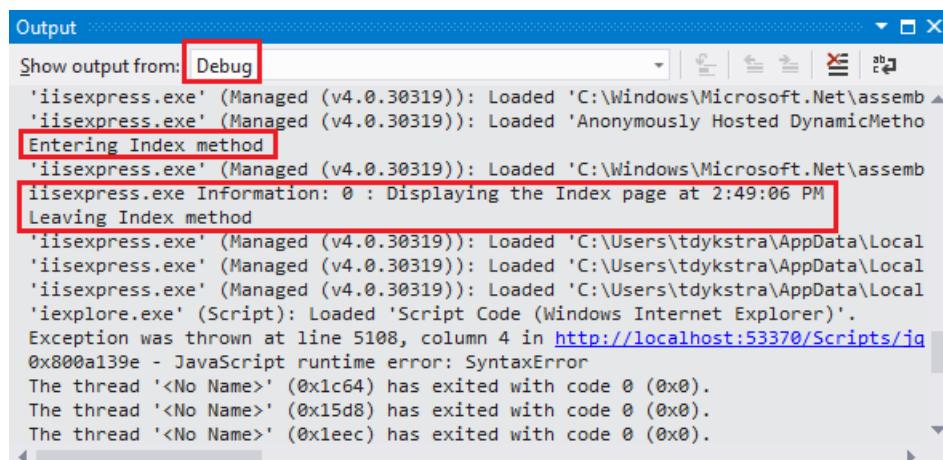
public ActionResult Contact()
{
 Trace.WriteLine("Entering Contact method");
 ViewBag.Message = "Your contact page.";
 Trace.TraceError("Fatal error on the Contact page at " + DateTime.Now.ToString());
 Trace.WriteLine("Leaving Contact method");
 return View();
}
```

2. Add a `using System.Diagnostics;` statement to the top of the file.

## View the tracing output locally

1. Press F5 to run the application in debug mode.

The default trace listener writes all trace output to the **Output** window, along with other Debug output. The following illustration shows the output from the trace statements that you added to the `Index` method.



The screenshot shows the Windows Task Manager with the 'Output' tab selected. The 'Show output from' dropdown is set to 'Debug'. The output window displays several lines of trace output:

```
iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System\2.0.0.0__b77a5c561934e089\System.dll'
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'Anonymously Hosted DynamicMethod Assembly'
[Red box highlights 'Entering Index method']
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Windows\Microsoft.Net\assembly\GAC_MSIL\System\2.0.0.0__b77a5c561934e089\System.dll'
[iisexpress.exe Information: 0 : Displaying the Index page at 2:49:06 PM]
[Red box highlights 'Leaving Index method']
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local\Temp\Temporary ASP.NET Files\root\12345678\12345678\Script1\'
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local\Temp\Temporary ASP.NET Files\root\12345678\12345678\Script1\'
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local\Temp\Temporary ASP.NET Files\root\12345678\12345678\Script1\'
'iexplore.exe' (Script): Loaded 'Script Code (Windows Internet Explorer)'.
Exception was thrown at line 5108, column 4 in http://localhost:53370/Scripts/jQuery.js
0x800a139e - JavaScript runtime error: SyntaxError
The thread '<No Name>' (0x1c64) has exited with code 0 (0x0).
The thread '<No Name>' (0x15d8) has exited with code 0 (0x0).
The thread '<No Name>' (0x1eec) has exited with code 0 (0x0).
```

The following steps show how to view trace output in a web page, without compiling in debug mode.

2. Open the application Web.config file (the one located in the project folder) and add a

```
<system.diagnostics> element at the end of the file just before the closing </configuration> element:
```

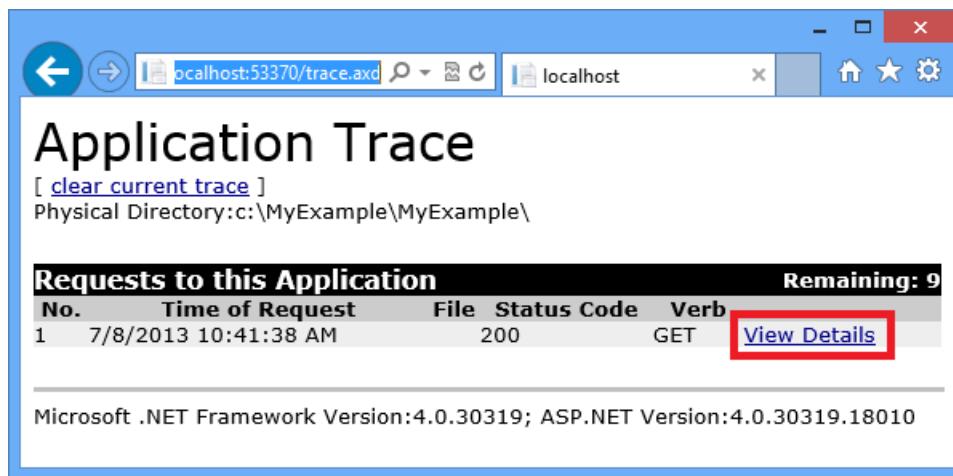
```
<system.diagnostics>
<trace>
 <listeners>
 <add name="WebPageTraceListener"
 type="System.Web.WebPageTraceListener,
 System.Web,
 Version=4.0.0.0,
 Culture=neutral,
 PublicKeyToken=b03f5f7f11d50a3a" />
 </listeners>
</trace>
</system.diagnostics>
```

The `WebPageTraceListener` lets you view trace output by browsing to `/trace.axd`.

1. Add a `trace` element under `<system.web>` in the Web.config file, such as the following example:

```
<trace enabled="true" writeToDiagnosticsTrace="true" mostRecent="true" pageOutput="false" />
```

2. Press CTRL+F5 to run the application.
3. In the address bar of the browser window, add `trace.axd` to the URL, and then press Enter (the URL is similar to `http://localhost:53370/trace.axd`).
4. On the Application Trace page, click **View Details** on the first line (not the BrowserLink line).



The Request Details page appears, and in the Trace Information section you see the output from the trace statements that you added to the `Index` method.

The screenshot shows a browser window with the URL `localhost:53370/Trace.axd?id=0`. The page title is "Request Details". The "Request Details" section contains the following information:

Session Id:	7/8/2013 10:41:38 AM	Request Type:	GET
Time of Request:	AM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

The "Trace Information" section displays a table of log entries:

Category	Message	From First (s)	From Last (s)
iisexpress.exe	Entering Index method Event 0: Displaying the Index page at 10:41:38 AM Leaving Index method	0.004651	0.004651
		0.004704	0.000054

A red box highlights the trace message "Entering Index method" and "Leaving Index method".

By default, `trace.axd` is only available locally. If you wanted to make it available from a remote app, you could add `localOnly="false"` to the `trace` element in the `Web.config` file, as shown in the following example:

```
<trace enabled="true" writeToDiagnosticsTrace="true" localOnly="false" mostRecent="true" pageOutput="false" />
```

However, enabling `trace.axd` in a production app is not recommended for security reasons. In the following sections, you'll see an easier way to read tracing logs in an App Service app.

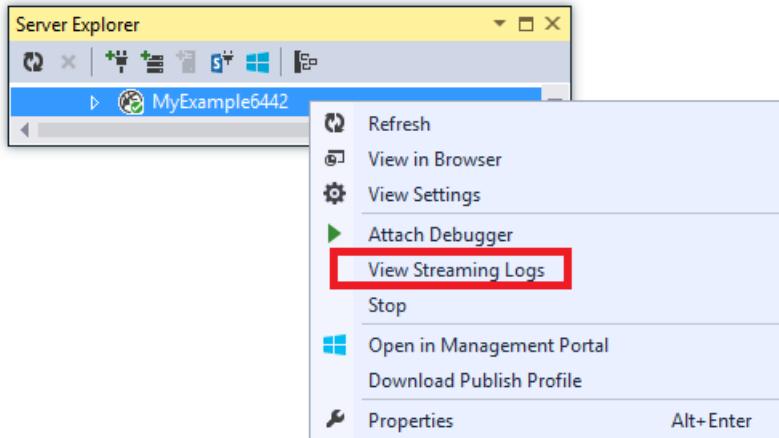
### View the tracing output in Azure

1. In Solution Explorer, right-click the web project and click **Publish**.

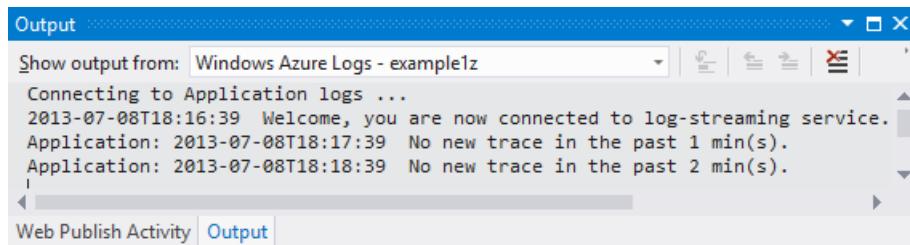
2. In the **Publish Web** dialog box, click **Publish**.

After Visual Studio publishes your update, it opens a browser window to your home page (assuming you didn't clear **Destination URL** on the **Connection** tab).

3. In Server Explorer, right-click your app and select **View Streaming Logs**.

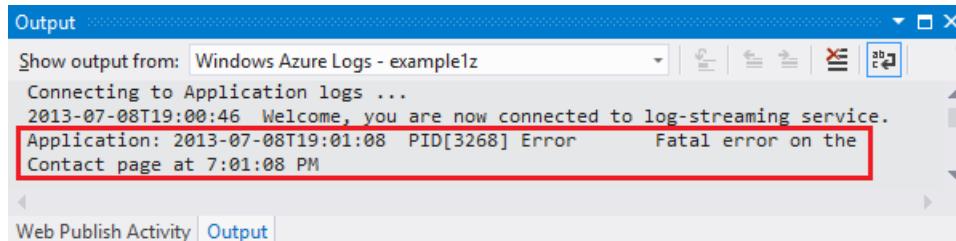


The Output window shows that you are connected to the log-streaming service, and adds a notification line each minute that goes by without a log to display.

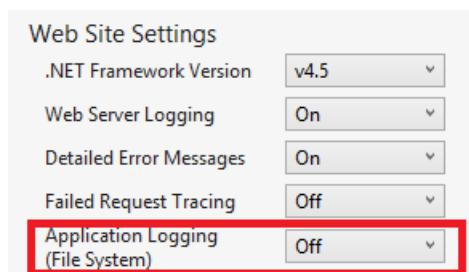


4. In the browser window that shows your application home page, click **Contact**.

Within a few seconds, the output from the error-level trace you added to the `Contact` method appears in the **Output** window.



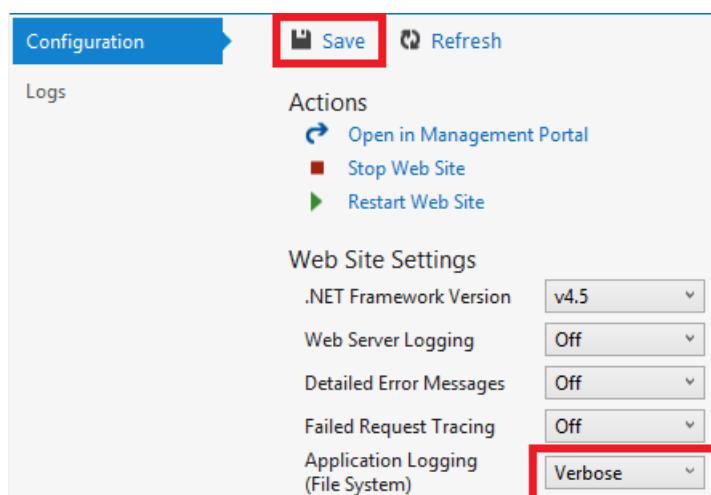
Visual Studio is only showing error-level traces because that is the default setting when you enable the log monitoring service. When you create a new App Service app, all logging is disabled by default, as you saw when you opened the settings page earlier:



However, when you selected **View Streaming Logs**, Visual Studio automatically changed **Application Logging(File System)** to **Error**, which means error-level logs get reported. In order to see all of your tracing logs, you can change this setting to **Verbose**. When you select a severity level lower than error, all logs for higher severity levels are also reported. So when you select verbose, you also see information, warning, and error logs.

5. In **Server Explorer**, right-click the app, and then click **View Settings** as you did earlier.

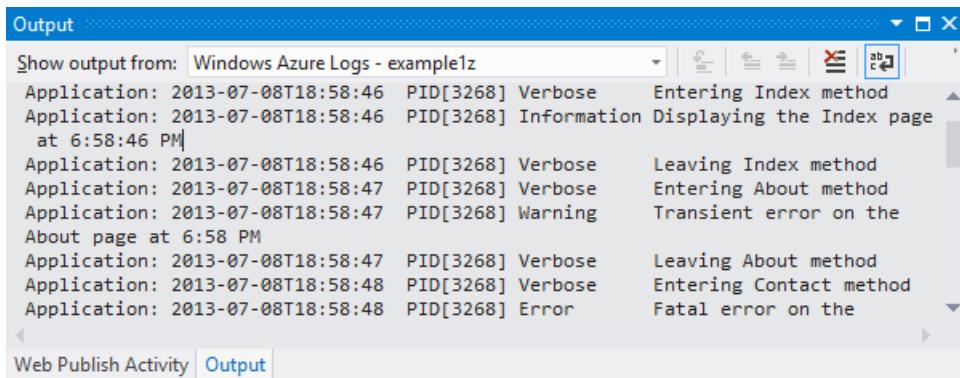
6. Change **Application Logging (File System)** to **Verbose**, and then click **Save**.



7. In the browser window that is now showing your **Contact** page, click **Home**, then click **About**, and then

click **Contact**.

Within a few seconds, the **Output** window shows all of your tracing output.



The screenshot shows the Microsoft Azure Logs tab in the Output window. The window title is "Output". The "Show output from:" dropdown is set to "Windows Azure Logs - example1z". The log entries are as follows:

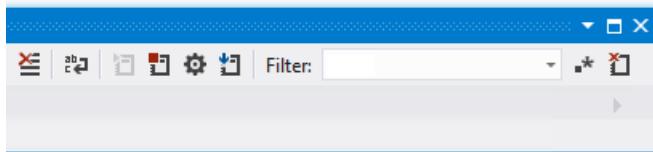
```
Application: 2013-07-08T18:58:46 PID[3268] Verbose Entering Index method
Application: 2013-07-08T18:58:46 PID[3268] Information Displaying the Index page
at 6:58:46 PM
Application: 2013-07-08T18:58:46 PID[3268] Verbose Leaving Index method
Application: 2013-07-08T18:58:47 PID[3268] Verbose Entering About method
Application: 2013-07-08T18:58:47 PID[3268] Warning Transient error on the
About page at 6:58 PM
Application: 2013-07-08T18:58:47 PID[3268] Verbose Leaving About method
Application: 2013-07-08T18:58:48 PID[3268] Verbose Entering Contact method
Application: 2013-07-08T18:58:48 PID[3268] Error Fatal error on the
```

At the bottom of the window, there are two tabs: "Web Publish Activity" and "Output". The "Output" tab is selected.

In this section, you enabled and disabled logging by using app settings. You can also enable and disable trace listeners by modifying the Web.config file. However, modifying the Web.config file causes the app domain to recycle, while enabling logging via the app configuration doesn't do that. If the problem takes a long time to reproduce, or is intermittent, recycling the app domain might "fix" it and force you to wait until it happens again. Enabling diagnostics in Azure lets you start capturing error information immediately without recycling the app domain.

### Output window features

The Microsoft Azure Logs tab of the Output Window has several buttons and a text box:



These perform the following functions:

- Clear the **Output** window.
- Enable or disable word wrap.
- Start or stop monitoring logs.
- Specify which logs to monitor.
- Download logs.
- Filter logs based on a search string or a regular expression.
- Close the **Output** window.

If you enter a search string or regular expression, Visual Studio filters logging information at the client. That means you can enter the criteria after the logs are displayed in the **Output** window and you can change filtering criteria without having to regenerate the logs.

## View web server logs

Web server logs record all HTTP activity for the app. In order to see them in the **Output** window, you must enable them for the app and tell Visual Studio that you want to monitor them.

1. In the **Azure Web App Configuration** tab that you opened from **Server Explorer**, change Web Server Logging to **On**, and then click **Save**.

Configuration

Save Refresh

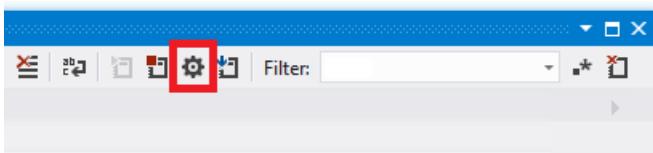
Logs Actions

- Open in Management Portal
- Stop Web Site
- Restart Web Site

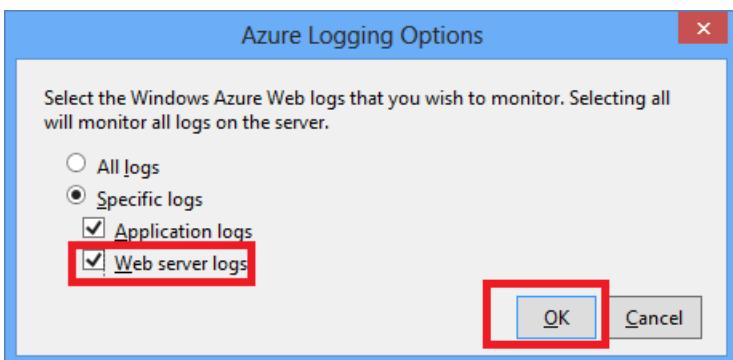
Web Site Settings

.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	Off
Failed Request Tracing	Off
Application Logging (File System)	Verbose

2. In the Output Window, click the Specify which Microsoft Azure logs to monitor button.



3. In the Microsoft Azure Logging Options dialog box, select Web server logs, and then click OK.



4. In the browser window that shows the app, click Home, then click About, and then click Contact.

The application logs generally appear first, followed by the web server logs. You might have to wait a while for the logs to appear.

Output

Show output from: Windows Azure Logs - example1z

```

Connecting to Application logs ...
2013-07-08T19:50:34 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:51:34 No new trace in the past 1 min(s).
Application: 2013-07-08T19:52:34 No new trace in the past 2 min(s).
Connecting to Web server logs ...
2013-07-08T19:52:36 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:53:34 No new trace in the past 3 min(s).
Web server: 2013-07-08T19:53:36 No new trace in the past 1 min(s).
Application: 2013-07-08T19:54:34 No new trace in the past 4 min(s).
Web server: 2013-07-08T19:54:36 No new trace in the past 2 min(s).
Application: 2013-07-08T19:55:06 PID[3268] Information DotNetOpenAuth.Core,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=2780ccd10d57b246 (official)
Application: 2013-07-08T19:55:06 PID[3268] Information Reporting will use
isolated storage with scope: Domain, Assembly, Machine
Application: 2013-07-08T19:55:10 PID[3268] Verbose Entering Contact method
Application: 2013-07-08T19:55:10 PID[3268] Verbose Entering Index method
Application: 2013-07-08T19:55:10 PID[3268] Verbose Entering About method
Application: 2013-07-08T19:55:10 PID[3268] Warning Transient error on the
About page at 7:55 PM
Application: 2013-07-08T19:55:10 PID[3268] Verbose Leaving About method
Application: 2013-07-08T19:55:10 PID[3268] Error Fatal error on the
Contact page at 7:55:10 PM
Application: 2013-07-08T19:55:10 PID[3268] Verbose Leaving Contact method
Application: 2013-07-08T19:55:10 PID[3268] Information Displaying the Index page
at 7:55:10 PM
Application: 2013-07-08T19:55:10 PID[3268] Verbose Leaving Index method
Web server: 2013-07-08T19:55:36 No new trace in the past 3 min(s).
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET /Home/Contact X-ARR-LOG-
ID=1f7e79df-bfbc-40c6-9573-cfc04612f93 80 - 131.107.0.112 Mozilla/5.0
+(compatible;+MSIE+10.0;+Windows+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 3307
623 7549
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET /Home/About X-ARR-LOG-
ID=8a35806d-8e92-479e-bad7-55d001c2fab2 80 - 131.107.0.112 Mozilla/5.0
+(compatible;+MSIE+10.0;+Windows+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 2845
619 8480
Web server: 2013-07-08 19:55:13 EXAMPLE1Z GET / X-ARR-LOG-ID=1e9c3f42-7f32-4a23-
a15b-037f2b69f870 80 - 131.107.0.112 Mozilla/5.0+(compatible;+MSIE+10.0;+Windows
+NT+6.2;+WOW64;+Trident/6.0)
ARRAffinity=948219045391acdeaa3c34903baa60b2db150a12c5d3f2589134a811deb4a38c;
+WAWebSiteSID=e0d53e3f499942b699075cd5f0881106 http://example1z.azurewebsites.net/Home/Contact example1z.azurewebsites.net 200 0 0 4137
599 9261
Application: 2013-07-08T19:56:34 No new trace in the past 1 min(s).
Application: 2013-07-08T19:57:34 No new trace in the past 2 min(s).
Web server: 2013-07-08T19:57:36 No new trace in the past 1 min(s).

```

Web Publish Activity | Output

By default, when you first enable web server logs by using Visual Studio, Azure writes the logs to the file system. As an alternative, you can use the Azure portal to specify that web server logs should be written to a blob container in a storage account.

If you use the portal to enable web server logging to an Azure storage account, and then disable logging in Visual Studio, when you re-enable logging in Visual Studio your storage account settings are restored.

## View detailed error message logs

Detailed error logs provide some additional information about HTTP requests that result in error response codes (400 or above). In order to see them in the **Output** window, you have to enable them for the app and tell Visual Studio that you want to monitor them.

1. In the **Azure Web App Configuration** tab that you opened from **Server Explorer**, change **Detailed Error Messages** to **On**, and then click **Save**.

Configuration

Save Refresh

Logs Actions

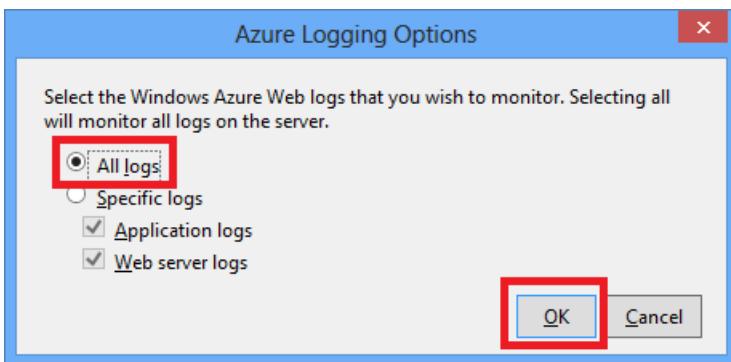
- Open in Management Portal
- Stop Web Site
- Restart Web Site

Web Site Settings

.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	On
Failed Request Tracing	Off
Application Logging (File System)	Verbose

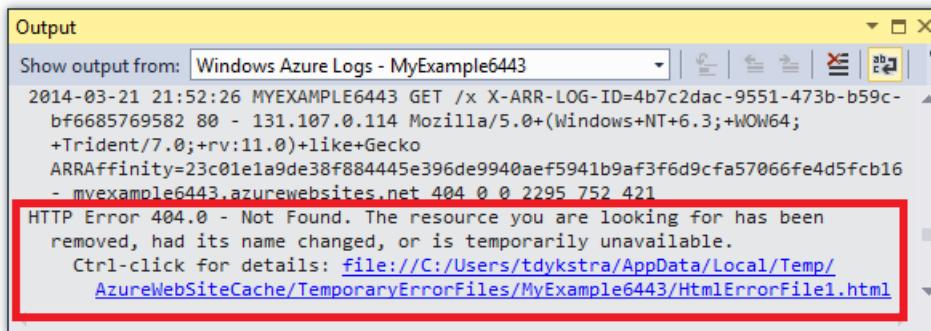
2. In the Output Window, click the Specify which Microsoft Azure logs to monitor button.

3. In the Microsoft Azure Logging Options dialog box, click All logs, and then click OK.



4. In the address bar of the browser window, add an extra character to the URL to cause a 404 error (for example, `http://localhost:53370/Home/Contactx`), and press Enter.

After several seconds, the detailed error log appears in the Visual Studio Output window.



Control+click the link to see the log output formatted in a browser:

The screenshot shows a browser window with the title "IIS Detailed Error - 404....". The main content is titled "HTTP Error 404.0 - Not Found" and contains the message: "The resource you are looking for has been removed, had its name changed, or is temporarily unavailable." Below this, a section titled "Most likely causes:" lists three items:

- The directory or file specified does not exist on the Web server.
- The URL contains a typographical error.
- A custom filter or module, such as URLScan, restricts access to the file.

Another section titled "Things you can try:" lists three items:

- Create the content on the Web server.
- Review the browser URL.
- Create a tracing rule to track failed requests for this HTTP status code and see which module is calling SetStatus. For more information about creating a tracing rule for failed requests, click [here](#).

A "Detailed Error Information:" section provides specific details:

Module	ManagedPipelineHandler	Request URL	http://example1z.azurewebsites.net:80/Home/Contactx
Notification	ExecuteRequestHandler	Physical Path	C:\DWASFiles\Sites\example1z\VirtualDirectory0\site\wwwroot\Home\Contactx
Handler	System.Web.Mvc.MvcHandler	Logon Method	Anonymous
Error Code	0x00000000	Logon User	Anonymous

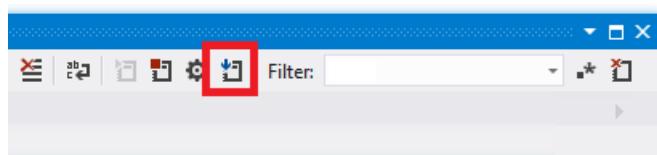
A "More Information:" section states: "This error means that the file or directory does not exist on the server. Create the file or directory and try the request again." It includes a link "[View more information >](#)".

Microsoft Knowledge Base Articles:

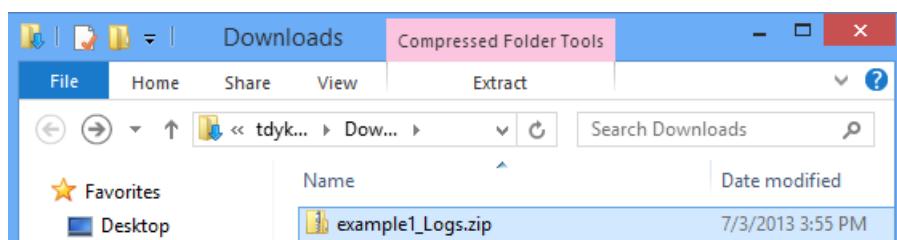
## Download file system logs

Any logs that you can monitor in the **Output** window can also be downloaded as a **.zip** file.

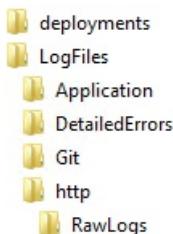
1. In the **Output** window, click **Download Streaming Logs**.



File Explorer opens to your *Downloads* folder with the downloaded file selected.



2. Extract the .zip file, and you see the following folder structure:



- Application tracing logs are in .txt files in the *LogFiles\Application* folder.
- Web server logs are in .log files in the *LogFiles\http\RawLogs* folder. You can use a tool such as [Log Parser](#) to view and manipulate these files.
- Detailed error message logs are in .html files in the *LogFiles\DetailedErrors* folder.

(The *deployments* folder is for files created by source control publishing; it doesn't have anything related to Visual Studio publishing. The *Git* folder is for traces related to source control publishing and the log file streaming service.)

## View failed request tracing logs

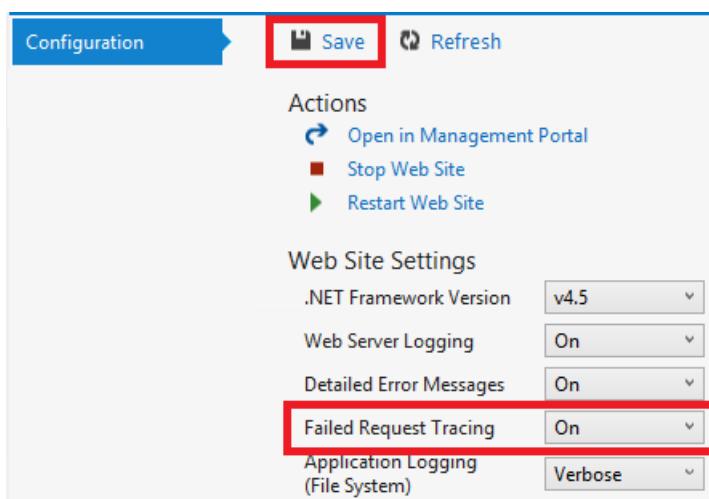
Failed request tracing logs are useful when you need to understand the details of how IIS is handling an HTTP request, in scenarios such as URL rewriting or authentication problems.

App Service apps use the same failed request tracing functionality that has been available with IIS 7.0 and later. You don't have access to the IIS settings that configure which errors get logged, however. When you enable failed request tracing, all errors are captured.

You can enable failed request tracing by using Visual Studio, but you can't view them in Visual Studio. These logs are XML files. The streaming log service only monitors files that are deemed readable in plain text mode: .txt, .html, and .log files.

You can view failed request tracing logs in a browser directly via FTP or locally after using an FTP tool to download them to your local computer. In this section, you'll view them in a browser directly.

1. In the **Configuration** tab of the Azure Web App window that you opened from **Server Explorer**, change **Failed Request Tracing** to **On**, and then click **Save**.



2. In the address bar of the browser window that shows the app, add an extra character to the URL and click Enter to cause a 404 error.

This causes a failed request tracing log to be created, and the following steps show how to view or download the log.

3. In Visual Studio, in the **Configuration** tab of the **Azure Web App** window, click **Open in Management Portal**.
4. In the [Azure portal](#) **Settings** page for your app, click **Deployment credentials**, and then enter a new user name and password.

**Search settings**

- Properties
- Application settings
- Scale
- Custom domains and SSL
- Deployment credentials**

**New name and password**

Git and FTP can't authenticate using the account you're signed in with, so create a new user name and password to use with those technologies

Use this user name and password to deploy to any site for all subscriptions associated with your Microsoft Azure account

FTP/deployment user name:

Password:

Confirm password:

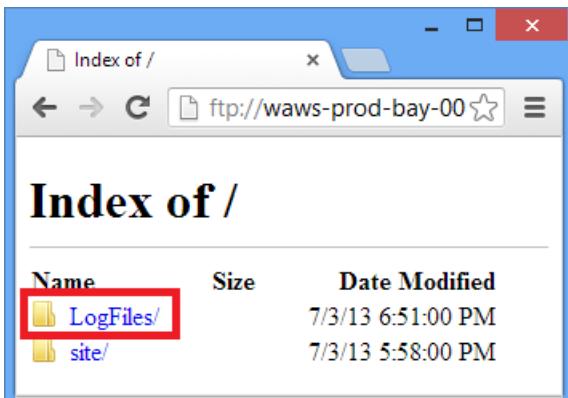
#### NOTE

When you log in, you have to use the full user name with the app name prefixed to it. For example, if you enter "myid" as a user name and the site is "myexample", you log in as "myexample\myid".

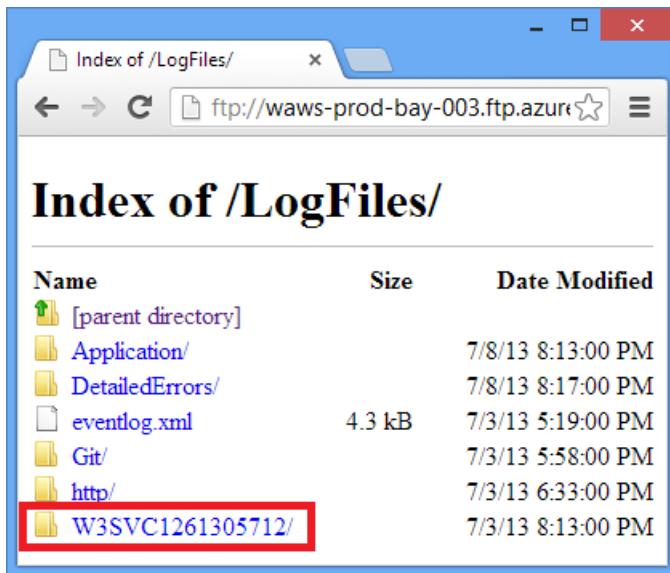
5. In a new browser window, go to the URL that is shown under **FTP hostname or FTPS hostname** in the **Overview** page for your app.
6. Sign in using the FTP credentials that you created earlier (including the app name prefix for the user name).

The browser shows the root folder of the app.

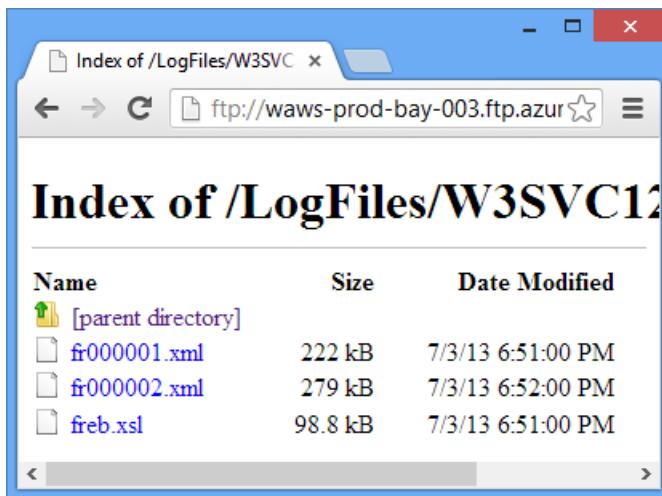
7. Open the *LogFiles* folder.



8. Open the folder that is named W3SVC plus a numeric value.



The folder contains XML files for any errors that have been logged after you enabled failed request tracing, and an XSL file that a browser can use to format the XML.



9. Click the XML file for the failed request that you want to see tracing information for.

The following illustration shows part of the tracing information for a sample error.

http://example1z.azurewebsites.net

ftp://waws-prod-bay-003.ftp.azurewebsites.windows.net/Logs

### Request Diagnostics for GET http://example1z.azurewebsites.net:80/Home/Contactx

**- Request Summary**

Site	1261305712
Process	3268
Failure Reason	STATUS_CODE
Trigger Status	404
Final Status	404
Time Taken	5125 msec

Url: http://example1z.azurewebsites.net:80/Home/Contactx  
 App Pool: example1z  
 Authentication: anonymous  
 User from token: IIS APPPOOL\example1z  
 Activity ID: {00000000-0000-0000-7369-0180000000F1}

---

**- Errors & Warnings**

No.	Severity	Event	Module Name
191.	view trace	Warning - MODULE_SET_RESPONSE_ERROR_STATUS	ManagedPipelineHandler
		ModuleName ManagedPipelineHandler	
		Notification EXECUTE_REQUEST_HANDLER	
		DLER	
		HttpStatus 404	
		HttpReason Not Found	
		HttpSubStatus 0	
		ErrorCode The operation completed successfully. (0x0)	
		ConfigExceptionInfo	

[See all events for the request](#)

---

No.	EventName	Details	Time
1.	GENERAL_REQUEST_START	SiteId="1261305712", AppPoolId="example1z", ConnId="1610705266", RawConnId="0", RequestURL="http://example1z.azurewebsites.net:80/Home/Contactx", RequestVerb="GET"	21:05:24.691
2.	PRE_BEGIN_REQUEST_START	ModuleName="FailedRequestsTracingModule"	21:05:24.722
3.	PRE_BEGIN_REQUEST_END	ModuleName="FailedRequestsTracingModule", NotificationStatus="NOTIFICATION_CONTINUE"	21:05:24.722
4.	PRE_BEGIN_REQUEST_START	ModuleName="RequestMonitorModule"	21:05:24.722
5.	PRE_BEGIN_REQUEST_END	ModuleName="RequestMonitorModule", NotificationStatus="NOTIFICATION_CONTINUE"	21:05:24.722
6.	PRE_BEGIN_REQUEST_START	ModuleName="IsapiFilterModule"	21:05:24.722
7.	FILTER_PREPROC_HEADERS_START		21:05:24.722
8.	FILTER_START	FilterName="D:\Windows\	21:05:24.722

## Next Steps

You've seen how Visual Studio makes it easy to view logs created by an App Service app. The following sections provide links to more resources on related topics:

- App Service troubleshooting
- Debugging in Visual Studio
- Remote debugging in Azure
- Tracing in ASP.NET applications

- Analyzing web server logs
- Analyzing failed request tracing logs
- Debugging Cloud Services

## App Service troubleshooting

For more information about troubleshooting apps in Azure App Service, see the following resources:

- [How to monitor apps](#)
- [Investigating Memory Leaks in Azure App Service with Visual Studio 2013](#). Microsoft ALM blog post about Visual Studio features for analyzing managed memory issues.
- [Azure App Service online tools you should know about](#). Blog post by Amit Apple.

For help with a specific troubleshooting question, start a thread in one of the following forums:

- [The Azure forum on the ASP.NET site](#).
- [The Azure forum on Microsoft Q&A](#).
- [StackOverflow.com](#).

## Debugging in Visual Studio

For more information about how to use debug mode in Visual Studio, see [Debugging in Visual Studio](#) and [Debugging Tips with Visual Studio 2010](#).

## Remote debugging in Azure

For more information about remote debugging for App Service apps and WebJobs, see the following resources:

- [Introduction to Remote Debugging Azure App Service](#).
- [Introduction to Remote Debugging Azure App Service part 2 - Inside Remote debugging](#)
- [Introduction to Remote Debugging on Azure App Service part 3 - Multi-Instance environment and GIT](#)
- [WebJobs Debugging \(video\)](#)

If your app uses an Azure Web API or Mobile Services back-end and you need to debug that, see [Debugging .NET Backend in Visual Studio](#).

## Tracing in ASP.NET applications

There are no thorough and up-to-date introductions to ASP.NET tracing available on the Internet. The best you can do is get started with old introductory materials written for Web Forms because MVC didn't exist yet, and supplement that with newer blog posts that focus on specific issues. Some good places to start are the following resources:

- [Monitoring and Telemetry \(Building Real-World Cloud Apps with Azure\)](#).  
E-book chapter with recommendations for tracing in Azure cloud applications.
- [ASP.NET Tracing](#)  
Old but still a good resource for a basic introduction to the subject.
- [Trace Listeners](#)  
Information about trace listeners but doesn't mention the [WebPageTraceListener](#).
- [Walkthrough: Integrating ASP.NET Tracing with System.Diagnostics Tracing](#)  
This article is also old, but includes some additional information that the introductory article doesn't cover.
- [Tracing in ASP.NET MVC Razor Views](#)  
Besides tracing in Razor views, the post also explains how to create an error filter in order to log all unhandled exceptions in an MVC application. For information about how to log all unhandled exceptions in a Web Forms application, see the Global.asax example in [Complete Example for Error Handlers](#) on

MSDN. In either MVC or Web Forms, if you want to log certain exceptions but let the default framework handling take effect for them, you can catch and rethrow as in the following example:

```
try
{
 // Your code that might cause an exception to be thrown.
}
catch (Exception ex)
{
 Trace.TraceError("Exception: " + ex.ToString());
 throw;
}
```

- [Streaming Diagnostics Trace Logging from the Azure Command Line \(plus Glimpse!\)](#)

How to use the command line to do what this tutorial shows how to do in Visual Studio. [Glimpse](#) is a tool for debugging ASP.NET applications.

- [Using Web Apps Logging and Diagnostics - with David Ebbo](#) and [Streaming Logs from Web Apps - with David Ebbo](#)

Videos by Scott Hanselman and David Ebbo.

For error logging, an alternative to writing your own tracing code is to use an open-source logging framework such as [ELMAH](#). For more information, see [Scott Hanselman's blog posts about ELMAH](#).

Also, you don't need to use ASP.NET or `System.Diagnostics` tracing to get streaming logs from Azure. The App Service app streaming log service streams any `.txt`, `.html`, or `.log` file that it finds in the `LogFiles` folder. Therefore, you could create your own logging system that writes to the file system of the app, and your file is automatically streamed and downloaded. All you have to do is write application code that creates files in the `d:\home\logfiles` folder.

## Analyzing web server logs

For more information about analyzing web server logs, see the following resources:

- [LogParser](#)  
A tool for viewing data in web server logs (`.log` files).
- [Troubleshooting IIS Performance Issues or Application Errors using LogParser](#)  
An introduction to the Log Parser tool that you can use to analyze web server logs.
- [Blog posts by Robert McMurray on using LogParser](#)
- [The HTTP status code in IIS 7.0, IIS 7.5, and IIS 8.0](#)

## Analyzing failed request tracing logs

The Microsoft TechNet website includes a [Using Failed Request Tracing](#) section, which may be helpful for understanding how to use these logs. However, this documentation focuses mainly on configuring failed request tracing in IIS, which you can't do in Azure App Service.

# Best practices and troubleshooting guide for node applications on Azure App Service Windows

11/2/2021 • 12 minutes to read • [Edit Online](#)

In this article, you learn best practices and troubleshooting steps for [Windows Node.js applications](#) running on Azure App Service (with [iisnode](#)).

## WARNING

Use caution when using troubleshooting steps on your production site. Recommendation is to troubleshoot your app on a non-production setup for example your staging slot and when the issue is fixed, swap your staging slot with your production slot.

## IISNODE configuration

This [schema file](#) shows all the settings that you can configure for iisnode. Some of the settings that are useful for your application:

### **nodeProcessCountPerApplication**

This setting controls the number of node processes that are launched per IIS application. The default value is 1. You can launch as many node.exe's as your VM vCPU count by changing the value to 0. The recommended value is 0 for most applications so you can use all of the vCPUs on your machine. Node.exe is single-threaded so one node.exe consumes a maximum of 1 vCPU. To get maximum performance out of your node application, you want to use all vCPUs.

### **nodeProcessCommandLine**

This setting controls the path to the node.exe. You can set this value to point to your node.exe version.

### **maxConcurrentRequestsPerProcess**

This setting controls the maximum number of concurrent requests sent by iisnode to each node.exe. On Azure App Service, the default value is Infinite. You can configure the value depending on how many requests your application receives and how fast your application processes each request.

### **maxNamedPipeConnectionRetry**

This setting controls the maximum number of times iisnode retries making the connection on the named pipe to send the requests to node.exe. This setting in combination with namedPipeConnectionRetryDelay determines the total timeout of each request within iisnode. The default value is 200 on Azure App Service. Total Timeout in seconds =  $(\text{maxNamedPipeConnectionRetry} * \text{namedPipeConnectionRetryDelay}) / 1000$

### **namedPipeConnectionRetryDelay**

This setting controls the amount of time (in ms) iisnode waits between each retry to send the request to node.exe over the named pipe. The default value is 250 ms. Total Timeout in seconds =  $(\text{maxNamedPipeConnectionRetry} * \text{namedPipeConnectionRetryDelay}) / 1000$

By default, the total timeout in iisnode on Azure App Service is  $200 * 250$  ms = 50 seconds.

### **logDirectory**

This setting controls the directory where iisnode logs stdout/stderr. The default value is iisnode, which is relative to the main script directory (directory where main server.js is present)

## **debuggerExtensionDll**

This setting controls what version of node-inspector iisnode uses when debugging your node application. Currently, iisnode-inspector-0.7.3.dll and iisnode-inspector.dll are the only two valid values for this setting. The default value is iisnode-inspector-0.7.3.dll. The iisnode-inspector-0.7.3.dll version uses node-inspector-0.7.3 and uses web sockets. Enable web sockets on your Azure webapp to use this version. See <https://ranjithblogs.azurewebsites.net/?p=98> for more details on how to configure iisnode to use the new node-inspector.

## **flushResponse**

The default behavior of IIS is that it buffers response data up to 4 MB before flushing, or until the end of the response, whichever comes first. iisnode offers a configuration setting to override this behavior: to flush a fragment of the response entity body as soon as iisnode receives it from node.exe, you need to set the iisnode/@flushResponse attribute in web.config to 'true':

```
<configuration>
 <system.webServer>
 <!-- ... -->
 <iisnode flushResponse="true" />
 </system.webServer>
</configuration>
```

Enable the flushing of every fragment of the response entity body adds performance overhead that reduces the throughput of the system by ~5% (as of v0.1.13). The best to scope this setting only to endpoints that require response streaming (for example, using the `<location>` element in the web.config)

In addition to this, for streaming applications, you must also set responseBufferLimit of your iisnode handler to 0.

```
<handlers>
 <add name="iisnode" path="app.js" verb="*" modules="iisnode" responseBufferLimit="0"/>
</handlers>
```

## **watchedFiles**

A semi-colon separated list of files that are watched for changes. Any change to a file causes the application to recycle. Each entry consists of an optional directory name as well as a required file name, which are relative to the directory where the main application entry point is located. Wild cards are allowed in the file name portion only. The default value is `*.js;iisnode.yml`

## **recycleSignalEnabled**

The default value is false. If enabled, your node application can connect to a named pipe (environment variable `IISNODE_CONTROL_PIPE`) and send a "recycle" message. This causes the w3wp to recycle gracefully.

## **idlePageOutTimePeriod**

The default value is 0, which means this feature is disabled. When set to some value greater than 0, iisnode will page out all its child processes every 'idlePageOutTimePeriod' in milliseconds. See [documentation](#) to understand what page out means. This setting is useful for applications that consume a high amount of memory and want to page out memory to disk occasionally to free up RAM.

### **WARNING**

Use caution when enabling the following configuration settings on production applications. The recommendation is to not enable them on live production applications.

## **debugHeaderEnabled**

The default value is false. If set to true, iisnode adds an HTTP response header `iisnode-debug` to every HTTP response it sends the `iisnode-debug` header value is a URL. Individual pieces of diagnostic information can be obtained by looking at the URL fragment, however, a visualization is available by opening the URL in a browser.

## **loggingEnabled**

This setting controls the logging of stdout and stderr by iisnode. iisnode captures stdout/stderr from node processes it launches and writes to the directory specified in the 'logDirectory' setting. Once this is enabled, your application writes logs to the file system and depending on the amount of logging done by the application, there could be performance implications.

## **devErrorsEnabled**

The default value is false. When set to true, iisnode displays the HTTP status code and Win32 error code on your browser. The win32 code is helpful in debugging certain types of issues.

## **debuggingEnabled (do not enable on live production site)**

This setting controls debugging feature. iisnode is integrated with node-inspector. By enabling this setting, you enable debugging of your node application. Upon enabling this setting, iisnode creates node-inspector files in 'debuggerVirtualDir' directory on the first debug request to your node application. You can load the node-inspector by sending a request to `http://yoursite/server.js/debug`. You can control the debug URL segment with 'debuggerPathSegment' setting. By default, debuggerPathSegment='debug'. You can set `debuggerPathSegment` to a GUID, for example, so that it is more difficult to be discovered by others.

Read [Debug node.js applications on Windows](#) for more details on debugging.

# Scenarios and recommendations/troubleshooting

## **My node application is making excessive outbound calls**

Many applications would want to make outbound connections as part of their regular operation. For example, when a request comes in, your node app would want to contact a REST API elsewhere and get some information to process the request. You would want to use a keep alive agent when making http or https calls. You could use the `agentkeepalive` module as your keep alive agent when making these outbound calls.

The `agentkeepalive` module ensures that sockets are reused on your Azure webapp VM. Creating a new socket on each outbound request adds overhead to your application. Having your application reuse sockets for outbound requests ensures that your application doesn't exceed the maxSockets that are allocated per VM. The recommendation on Azure App Service is to set the `agentKeepAlive maxSockets` value to a total of (4 instances of `node.exe` \* 32 maxSockets/`instance`) 128 sockets per VM.

Example `agentKeepAlive` configuration:

```
let keepaliveAgent = new Agent({
 maxSockets: 32,
 maxFreeSockets: 10,
 timeout: 60000,
 keepAliveTimeout: 300000
});
```

### **IMPORTANT**

This example assumes you have 4 `node.exe` running on your VM. If you have a different number of `node.exe` running on the VM, you must modify the `maxSockets` setting accordingly.

## **My node application is consuming too much CPU**

You may receive a recommendation from Azure App Service on your portal about high cpu consumption. You can also set up monitors to watch for certain [metrics](#). When checking the CPU usage on the [Azure portal Dashboard](#), check the MAX values for CPU so you don't miss the peak values. If you believe your application is consuming too much CPU and you cannot explain why, you can profile your node application to find out.

#### Profiling your node application on Azure App Service with V8-Profiler

For example, let's say you have a hello world app that you want to profile as follows:

```
const http = require('http');
function WriteConsoleLog() {
 for(let i=0;i<99999;++i) {
 console.log('hello world');
 }
}

function HandleRequest() {
 WriteConsoleLog();
}

http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'});
 HandleRequest();
 res.end('Hello world!');
}).listen(process.env.PORT);
```

Go to the Debug Console site <https://yoursite.scm.azurewebsites.net/DebugConsole>

Go into your site/wwwroot directory. You see a command prompt as shown in the following example:

The screenshot shows the Kudu Remote Execution Console interface. At the top, there is a file browser showing the contents of the wwwroot directory:

	Name	Modified	Size
📄	node_modules	5/25/2016, 12:53:41 PM	
📄	hostingstart.html	5/25/2016, 12:51:16 PM	198 KB
📄	server.js	5/25/2016, 12:57:09 PM	1 KB
📄	web.config	5/25/2016, 12:52:34 PM	1 KB

Below the file browser is a command prompt window. It displays the following text:

```
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new CMD process.
Type 'cls' to clear the console

Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

D:\home>
D:\home\site>
D:\home\site\wwwroot>npm install v8-profiler
```

Run the command `npm install v8-profiler`.

This command installs the v8-profiler under node\_modules directory and all of its dependencies. Now, edit your serverjs to profile your application.

```

const http = require('http');
const profiler = require('v8-profiler');
const fs = require('fs');

function WriteConsoleLog() {
 for(let i=0;i<99999;++i) {
 console.log('hello world');
 }
}

function HandleRequest() {
 profiler.startProfiling('HandleRequest');
 WriteConsoleLog();
 fs.writeFileSync('profile.cpuprofile', JSON.stringify(profiler.stopProfiling('HandleRequest')));
}

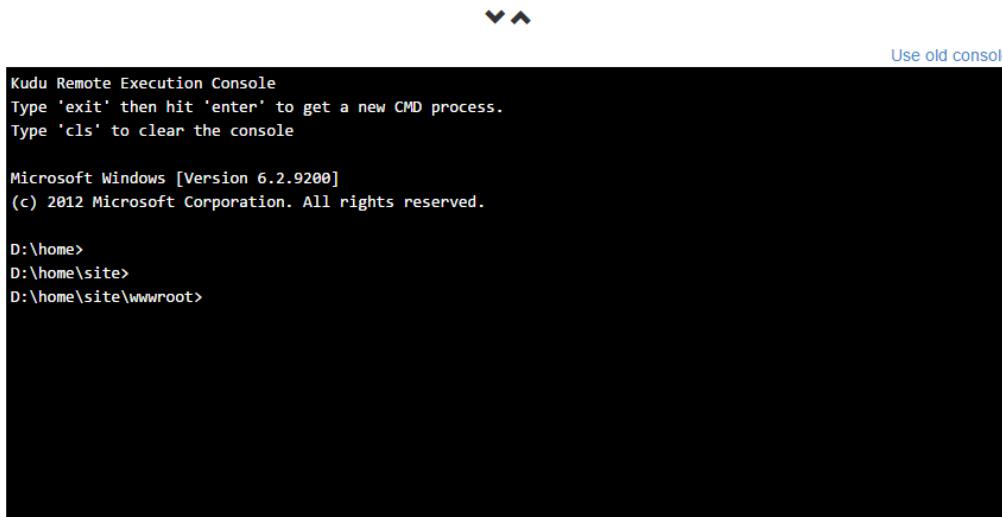
http.createServer(function (req, res) {
 res.writeHead(200, {'Content-Type': 'text/html'});
 HandleRequest();
 res.end('Hello world!');
}).listen(process.env.PORT);

```

The preceding code profiles the WriteConsoleLog function and then writes the profile output to the 'profile.cpuprofile' file under your site wwwroot. Send a request to your application. You see a 'profile.cpuprofile' file created under your site wwwroot.

... / wwwroot + | 5 items   

	Name	Modified	Size
  	node_modules	5/25/2016, 12:53:41 PM	
  	hostingstart.html	5/25/2016, 12:51:16 PM	198 KB
  	profile.cpuprofile	5/25/2016, 1:05:18 PM	3 KB
  	server.js	5/25/2016, 12:57:09 PM	1 KB
  	web.config	5/25/2016, 12:52:34 PM	1 KB

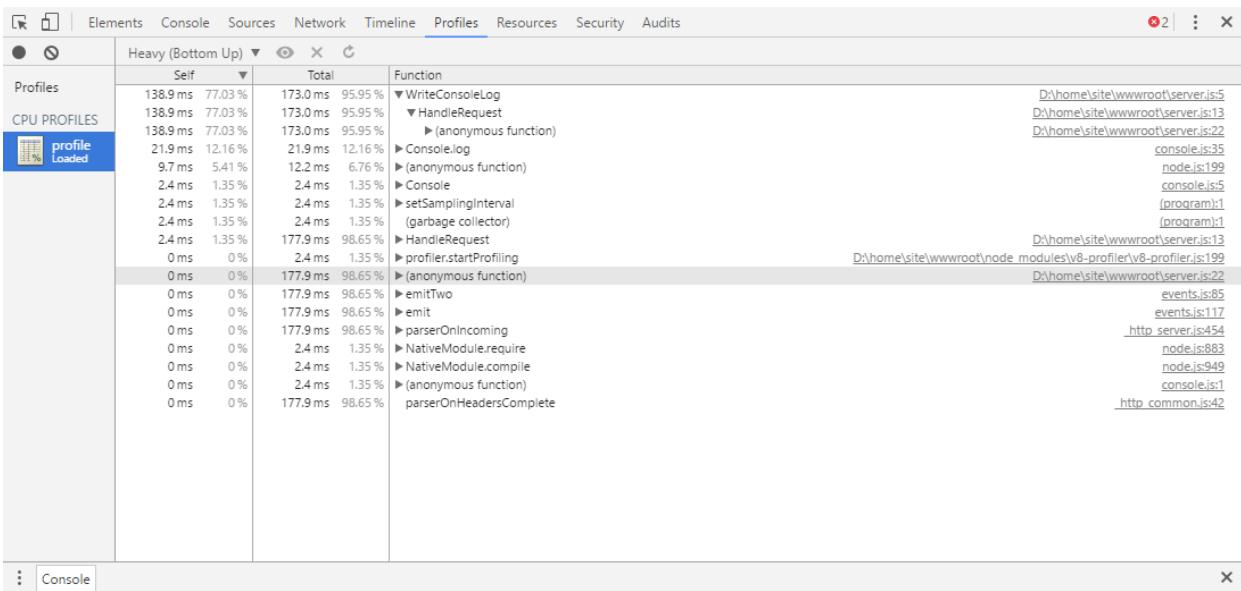


Kudu Remote Execution Console  
Type 'exit' then hit 'enter' to get a new CMD process.  
Type 'cls' to clear the console

Microsoft Windows [Version 6.2.9200]  
(c) 2012 Microsoft Corporation. All rights reserved.

D:\home>  
D:\home\site>  
D:\home\site\wwwroot>

Download this file and open it with Chrome F12 Tools. Press F12 on Chrome, then choose the **Profiles** tab. Choose the **Load** button. Select your profile.cpuprofile file that you downloaded. Click on the profile you just loaded.



You can see that 95% of the time was consumed by the WriteConsoleLog function. The output also shows you the exact line numbers and source files that caused the issue.

### My node application is consuming too much memory

If your application is consuming too much memory, you see a notice from Azure App Service on your portal about high memory consumption. You can set up monitors to watch for certain [metrics](#). When checking the memory usage on the [Azure portal Dashboard](#), be sure to check the MAX values for memory so you don't miss the peak values.

#### Leak detection and Heap Diff for node.js

You could use [node-memwatch](#) to help you identify memory leaks. You can install `memwatch` just like v8-profiler and edit your code to capture and diff heaps to identify the memory leaks in your application.

### My node.exe's are getting killed randomly

There are a few reasons why node.exe is shut down randomly:

1. Your application is throwing uncaught exceptions – Check d:\home\LogFiles\Application\logging-errors.txt file for the details on the exception thrown. This file has the stack trace to help debug and fix your application.
2. Your application is consuming too much memory, which is affecting other processes from getting started. If the total VM memory is close to 100%, your node.exe's could be killed by the process manager. Process manager kills some processes to let other processes get a chance to do some work. To fix this issue, profile your application for memory leaks. If your application requires large amounts of memory, scale up to a larger VM (which increases the RAM available to the VM).

### My node application does not start

If your application is returning 500 Errors when it starts, there could be a few reasons:

1. Node.exe is not present at the correct location. Check nodeProcessCommandLine setting.
2. Main script file is not present at the correct location. Check web.config and make sure the name of the main script file in the handlers section matches the main script file.
3. Web.config configuration is not correct – check the settings names/values.
4. Cold Start – Your application is taking too long to start. If your application takes longer than  $(\text{maxNamedPipeConnectionRetry} * \text{namedPipeConnectionRetryDelay}) / 1000$  seconds, iisnode returns a 500 error. Increase the values of these settings to match your application start time to prevent iisnode from timing out and returning the 500 error.

### My node application crashed

Your application is throwing uncaught exceptions – Check `d:\home\LogFiles\Application\logging-errors.txt`

file for the details on the exception thrown. This file has the stack trace to help diagnose and fix your application.

### My node application takes too much time to start (Cold Start)

The common cause for long application start times is a high number of files in the node\_modules. The application tries to load most of these files when starting. By default, since your files are stored on the network share on Azure App Service, loading many files can take time. Some solutions to make this process faster are:

1. Try to lazy load your node\_modules and not load all of the modules at application start. To Lazy load modules, the call to require('module') should be made when you actually need the module within the function before the first execution of module code.
2. Azure App Service offers a feature called local cache. This feature copies your content from the network share to the local disk on the VM. Since the files are local, the load time of node\_modules is much faster.

## IISNODE http status and substatus

The [cnodeconstants](#) [source file](#) lists all of the possible status/substatus combinations iisnode can return due to an error.

Enable FREB for your application to see the win32 error code (be sure you enable FREB only on non-production sites for performance reasons).

HTTP STATUS	HTTP SUBSTATUS	POSSIBLE REASON?
500	1000	There was some issue dispatching the request to IISNODE – Check if node.exe was started. Node.exe could have crashed when starting. Check your web.config configuration for errors.
500	1001	- Win32Error 0x2 - App is not responding to the URL. Check the URL rewrite rules or check if your express app has the correct routes defined. - Win32Error 0x6d – named pipe is busy – Node.exe is not accepting requests because the pipe is busy. Check high cpu usage. - Other errors – check if node.exe crashed.
500	1002	Node.exe crashed – check d:\home\LogFiles\logging-errors.txt for stack trace.
500	1003	Pipe configuration Issue – The named pipe configuration is incorrect.
500	1004-1018	There was some error while sending the request or processing the response to/from node.exe. Check if node.exe crashed. check d:\home\LogFiles\logging-errors.txt for stack trace.

HTTP STATUS	HTTP SUBSTATUS	POSSIBLE REASON?
503	1000	Not enough memory to allocate more named pipe connections. Check why your app is consuming so much memory. Check <code>maxConcurrentRequestsPerProcess</code> setting value. If it's not infinite and you have many requests, increase this value to prevent this error.
503	1001	Request could not be dispatched to node.exe because the application is recycling. After the application has recycled, requests should be served normally.
503	1002	Check win32 error code for actual reason – Request could not be dispatched to a node.exe.
503	1003	Named pipe is too Busy – Verify if node.exe is consuming excessive CPU

NODE.exe has a setting called `NODE_PENDING_PIPE_INSTANCES`. On Azure App Service, this value is set to 5000. Meaning that node.exe can accept 5000 requests at a time on the named pipe. This value should be good enough for most node applications running on Azure App Service. You should not see 503.1003 on Azure App Service because of the high value for the `NODE_PENDING_PIPE_INSTANCES`

## More resources

Follow these links to learn more about node.js applications on Azure App Service.

- [Get started with Node.js web apps in Azure App Service](#)
- [How to debug a Node.js web app in Azure App Service](#)
- [Using Node.js Modules with Azure applications](#)
- [Azure App Service Web Apps: Node.js](#)
- [Node.js Developer Center](#)
- [Exploring the Super Secret Kudu Debug Console](#)

# Troubleshoot HTTP errors of "502 bad gateway" and "503 service unavailable" in Azure App Service

11/2/2021 • 4 minutes to read • [Edit Online](#)

"502 bad gateway" and "503 service unavailable" are common errors in your app hosted in [Azure App Service](#). This article helps you troubleshoot these errors.

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and the Stack Overflow forums](#). Alternatively, you can also file an Azure support incident. Go to the [Azure Support site](#) and click on **Get Support**.

## Symptom

When you browse to the app, it returns a HTTP "502 Bad Gateway" error or a HTTP "503 Service Unavailable" error.

## Cause

This problem is often caused by application level issues, such as:

- requests taking a long time
- application using high memory/CPU
- application crashing due to an exception.

## Troubleshooting steps to solve "502 bad gateway" and "503 service unavailable" errors

Troubleshooting can be divided into three distinct tasks, in sequential order:

1. [Observe and monitor application behavior](#)
2. [Collect data](#)
3. [Mitigate the issue](#)

[App Service](#) gives you various options at each step.

### 1. Observe and monitor application behavior

#### Track Service health

Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the [Azure Portal](#). For more information, see [Track service health](#).

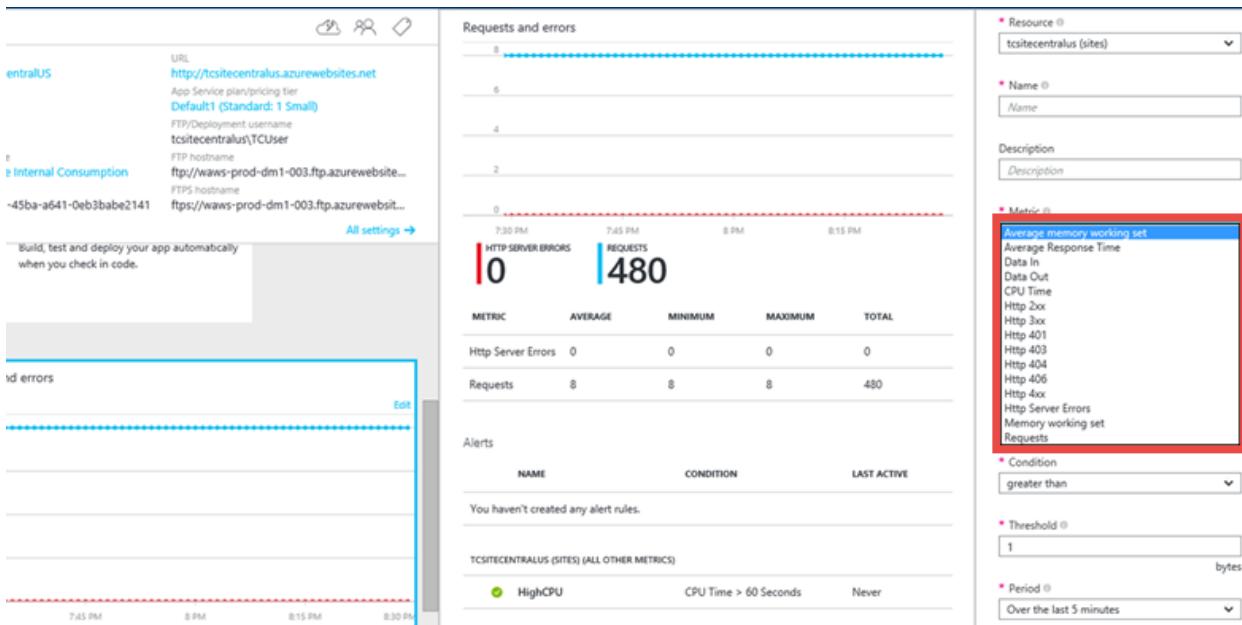
#### Monitor your app

This option enables you to find out if your application is having any issues. In your app's blade, click the **Requests and errors** tile. The **Metric** blade will show you all the metrics you can add.

Some of the metrics that you might want to monitor for your app are

- Average memory working set
- Average response time
- CPU time
- Memory working set

- Requests



For more information, see:

- [Monitor apps in Azure App Service](#)
- [Receive alert notifications](#)

## 2. Collect data

### Use the diagnostics tool

App Service provides an intelligent and interactive experience to help you troubleshoot your app with no configuration required. When you do run into issues with your app, the diagnostics tool will point out what's wrong to guide you to the right information to more easily and quickly troubleshoot and resolve the issue.

To access App Service diagnostics, navigate to your App Service app or App Service Environment in the [Azure portal](#). In the left navigation, click on **Diagnose and solve problems**.

### Use the Kudu Debug Console

App Service comes with a debug console that you can use for debugging, exploring, uploading files, as well as JSON endpoints for getting information about your environment. This is called the *Kudu Console* or the *SCM Dashboard* for your app.

You can access this dashboard by going to the link <https://<Your app name>.scm.azurewebsites.net/>.

Some of the things that Kudu provides are:

- environment settings for your application
- log stream
- diagnostic dump
- debug console in which you can run Powershell cmdlets and basic DOS commands.

Another useful feature of Kudu is that, in case your application is throwing first-chance exceptions, you can use Kudu and the SysInternals tool ProcDump to create memory dumps. These memory dumps are snapshots of the process and can often help you troubleshoot more complicated issues with your app.

For more information on features available in Kudu, see [Azure Websites online tools you should know about](#).

## 3. Mitigate the issue

### Scale the app

In Azure App Service, for increased performance and throughput, you can adjust the scale at which you are

running your application. Scaling up an app involves two related actions: changing your App Service plan to a higher pricing tier, and configuring certain settings after you have switched to the higher pricing tier.

For more information on scaling, see [Scale an app in Azure App Service](#).

Additionally, you can choose to run your application on more than one instance . This not only provides you with more processing capability, but also gives you some amount of fault tolerance. If the process goes down on one instance, the other instance will still continue serving requests.

You can set the scaling to be Manual or Automatic.

#### Use AutoHeal

AutoHeal recycles the worker process for your app based on settings you choose (like configuration changes, requests, memory-based limits, or the time needed to execute a request). Most of the time, recycle the process is the fastest way to recover from a problem. Though you can always restart the app from directly within the Azure Portal, AutoHeal will do it automatically for you. All you need to do is add some triggers in the root web.config for your app. Note that these settings would work in the same way even if your application is not a .NET one.

For more information, see [Auto-Healing Azure Web Sites](#).

#### Restart the app

This is often the simplest way to recover from one-time issues. On the [Azure Portal](#), on your app's blade, you have the options to stop or restart your app.



You can also manage your app using Azure Powershell. For more information, see [Using Azure PowerShell with Azure Resource Manager](#).

# Troubleshoot slow app performance issues in Azure App Service

11/2/2021 • 8 minutes to read • [Edit Online](#)

This article helps you troubleshoot slow app performance issues in [Azure App Service](#).

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN Azure and the Stack Overflow forums](#). Alternatively, you can also file an Azure support incident. Go to the [Azure Support site](#) and click on **Get Support**.

## Symptom

When you browse the app, the pages load slowly and sometimes timeout.

## Cause

This problem is often caused by application level issues, such as:

- network requests taking a long time
- application code or database queries being inefficient
- application using high memory/CPU
- application crashing due to an exception

## Troubleshooting steps

Troubleshooting can be divided into three distinct tasks, in sequential order:

1. [Observe and monitor application behavior](#)
2. [Collect data](#)
3. [Mitigate the issue](#)

[App Service](#) gives you various options at each step.

### 1. Observe and monitor application behavior

#### Track Service health

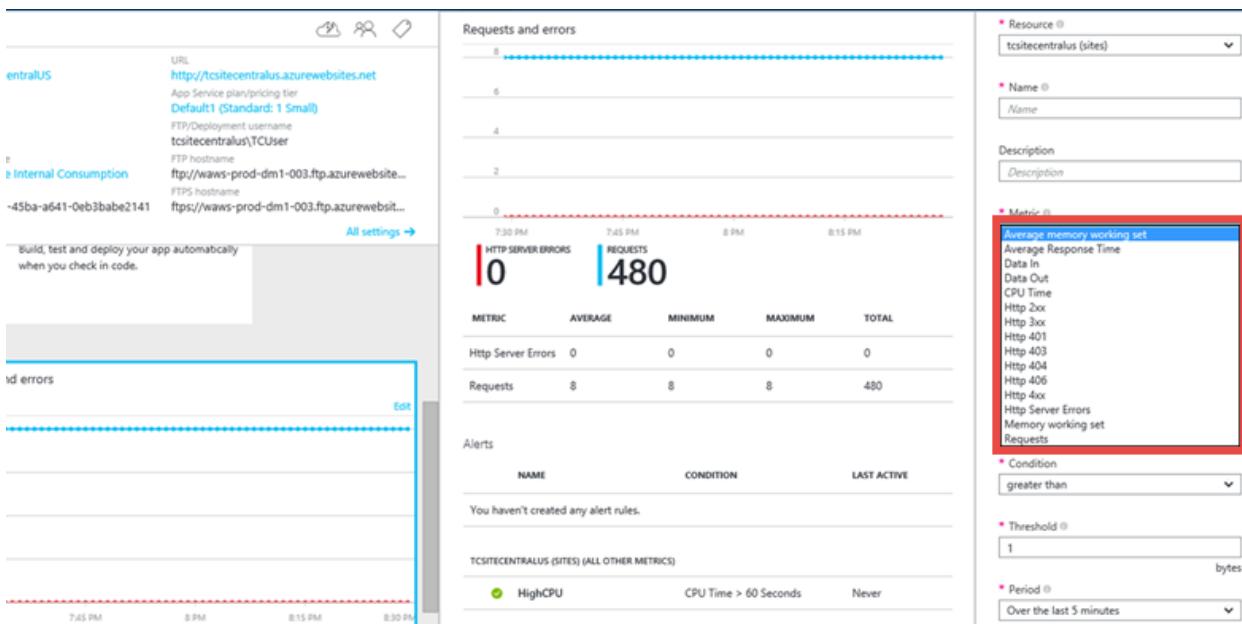
Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the [Azure portal](#). For more information, see [Track service health](#).

#### Monitor your app

This option enables you to find out if your application is having any issues. In your app's blade, click the **Requests and errors** tile. The **Metric** blade shows you all the metrics you can add.

Some of the metrics that you might want to monitor for your app are

- Average memory working set
- Average response time
- CPU time
- Memory working set
- Requests



For more information, see:

- [Monitor apps in Azure App Service](#)
- [Receive alert notifications](#)

#### Monitor web endpoint status

If you are running your app in the **Standard** pricing tier, App Service lets you monitor two endpoints from three geographic locations.

Endpoint monitoring configures web tests from geo-distributed locations that test response time and uptime of web URLs. The test performs an HTTP GET operation on the web URL to determine the response time and uptime from each location. Each configured location runs a test every five minutes.

Uptime is monitored using HTTP response codes, and response time is measured in milliseconds. A monitoring test fails if the HTTP response code is greater than or equal to 400 or if the response takes more than 30 seconds. An endpoint is considered available if its monitoring tests succeed from all the specified locations.

To set it up, see [Monitor apps in Azure App Service](#).

Also, see [Keeping Azure Web Sites up plus Endpoint Monitoring - with Stefan Schackow](#) for a video on endpoint monitoring.

#### Application performance monitoring using Extensions

You can also monitor your application performance by using a *site extension*.

Each App Service app provides an extensible management end point that allows you to use a powerful set of tools deployed as site extensions. Extensions include:

- Source code editors like [Azure DevOps](#).
- Management tools for connected resources such as a MySQL database connected to an app.

[Azure Application Insights](#) is a performance monitoring site extension that's also available. To use Application Insights, you rebuild your code with an SDK. You can also install an extension that provides access to additional data. The SDK lets you write code to monitor the usage and performance of your app in more detail. For more information, see [Monitor performance in web applications](#).

## 2. Collect data

App Service provides diagnostic functionality for logging information from both the web server and the web application. The information is separated into web server diagnostics and application diagnostics.

## **Enable web server diagnostics**

You can enable or disable the following kinds of logs:

- **Detailed Error Logging** - Detailed error information for HTTP status codes that indicate a failure (status code 400 or greater). This may contain information that can help determine why the server returned the error code.
- **Failed Request Tracing** - Detailed information on failed requests, including a trace of the IIS components used to process the request and the time taken in each component. This can be useful if you are attempting to improve app performance or isolate what is causing a specific HTTP error.
- **Web Server Logging** - Information about HTTP transactions using the W3C extended log file format. This is useful when determining overall app metrics, such as the number of requests handled or how many requests are from a specific IP address.

## **Enable application diagnostics**

There are several options to collect application performance data from App Service, profile your application live from Visual Studio, or modify your application code to log more information and traces. You can choose the options based on how much access you have to the application and what you observed from the monitoring tools.

### **Use Application Insights Profiler**

You can enable the Application Insights Profiler to start capturing detailed performance traces. You can access traces captured up to five days ago when you need to investigate problems happened in the past. You can choose this option as long as you have access to the app's Application Insights resource on Azure portal.

Application Insights Profiler provides statistics on response time for each web call and traces that indicates which line of code caused the slow responses. Sometimes the App Service app is slow because certain code is not written in a performant way. Examples include sequential code that can be run in parallel and undesired database lock contentions. Removing these bottlenecks in the code increases the app's performance, but they are hard to detect without setting up elaborate traces and logs. The traces collected by Application Insights Profiler helps identifying the lines of code that slows down the application and overcome this challenge for App Service apps.

For more information, see [Profiling live apps in Azure App Service with Application Insights](#).

### **Use Remote Profiling**

In Azure App Service, web apps, API apps, mobile back ends, and WebJobs can be remotely profiled. Choose this option if you have access to the app resource and you know how to reproduce the issue, or if you know the exact time interval the performance issue happens.

Remote Profiling is useful if the CPU usage of the process is high and your process is running slower than expected, or the latency of HTTP requests are higher than normal, you can remotely profile your process and get the CPU sampling call stacks to analyze the process activity and code hot paths.

For more information, see [Remote Profiling support in Azure App Service](#).

### **Set up diagnostic traces manually**

If you have access to the web application source code, Application diagnostics enables you to capture information produced by a web application. ASP.NET applications can use the `System.Diagnostics.Trace` class to log information to the application diagnostics log. However, you need to change the code and redeploy your application. This method is recommended if your app is running on a testing environment.

For detailed instructions on how to configure your application for logging, see [Enable diagnostics logging for apps in Azure App Service](#).

## **Use the diagnostics tool**

App Service provides an intelligent and interactive experience to help you troubleshoot your app with no configuration required. When you do run into issues with your app, the diagnostics tool will point out what's wrong to guide you to the right information to more easily and quickly troubleshoot and resolve the issue.

To access App Service diagnostics, navigate to your App Service app or App Service Environment in the [Azure portal](#). In the left navigation, click on **Diagnose and solve problems**.

#### Use the Kudu Debug Console

App Service comes with a debug console that you can use for debugging, exploring, uploading files, as well as JSON endpoints for getting information about your environment. This console is called the *Kudu Console* or the *SCM Dashboard* for your app.

You can access this dashboard by going to the link <https://<Your app name>.scm.azurewebsites.net/>.

Some of the things that Kudu provides are:

- environment settings for your application
- log stream
- diagnostic dump
- debug console in which you can run PowerShell cmdlets and basic DOS commands.

Another useful feature of Kudu is that, in case your application is throwing first-chance exceptions, you can use Kudu and the SysInternals tool ProcDump to create memory dumps. These memory dumps are snapshots of the process and can often help you troubleshoot more complicated issues with your app.

For more information on features available in Kudu, see [Azure DevOps tools you should know about](#).

### 3. Mitigate the issue

#### Scale the app

In Azure App Service, for increased performance and throughput, you can adjust the scale at which you are running your application. Scaling up an app involves two related actions: changing your App Service plan to a higher pricing tier, and configuring certain settings after you have switched to the higher pricing tier.

For more information on scaling, see [Scale an app in Azure App Service](#).

Additionally, you can choose to run your application on more than one instance. Scaling out not only provides you with more processing capability, but also gives you some amount of fault tolerance. If the process goes down on one instance, the other instances continue to serve requests.

You can set the scaling to be Manual or Automatic.

#### Use AutoHeal

AutoHeal recycles the worker process for your app based on settings you choose (like configuration changes, requests, memory-based limits, or the time needed to execute a request). Most of the time, recycle the process is the fastest way to recover from a problem. Though you can always restart the app from directly within the Azure portal, AutoHeal does it automatically for you. All you need to do is add some triggers in the root web.config for your app. These settings would work in the same way even if your application is not a .NET app.

For more information, see [Auto-Healing Azure Web Sites](#).

#### Restart the app

Restarting is often the simplest way to recover from one-time issues. On the [Azure portal](#), on your app's blade, you have the options to stop or restart your app.



You can also manage your app using Azure PowerShell. For more information, see [Using Azure PowerShell with Azure Resource Manager](#).

# Troubleshoot domain and TLS/SSL certificate problems in Azure App Service

11/2/2021 • 13 minutes to read • [Edit Online](#)

This article lists common problems that you might encounter when you configure a domain or TLS/SSL certificate for your web apps in Azure App Service. It also describes possible causes and solutions for these problems.

If you need more help at any point in this article, you can contact the Azure experts on [the MSDN and Stack Overflow forums](#). Alternatively, you can file an Azure support incident. Go to the [Azure Support site](#) and select **Get Support**.

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Certificate problems

### You can't add a TLS/SSL certificate binding to an app

#### Symptom

When you add a TLS binding, you receive the following error message:

"Failed to add SSL binding. Cannot set certificate for existing VIP because another VIP already uses that certificate."

#### Cause

This problem can occur if you have multiple IP-based TLS/SSL bindings for the same IP address across multiple apps. For example, app A has an IP-based TLS/SSL binding with an old certificate. App B has an IP-based TLS/SSL binding with a new certificate for the same IP address. When you update the app TLS binding with the new certificate, it fails with this error because the same IP address is being used for another app.

#### Solution

To fix this problem, use one of the following methods:

- Delete the IP-based TLS/SSL binding on the app that uses the old certificate.
- Create a new IP-based TLS/SSL binding that uses the new certificate.

### You can't delete a certificate

#### Symptom

When you try to delete a certificate, you receive the following error message:

"Unable to delete the certificate because it is currently being used in a TLS/SSL binding. The TLS binding must be removed before you can delete the certificate."

#### Cause

This problem might occur if another app uses the certificate.

#### Solution

Remove the TLS binding for that certificate from the apps. Then try to delete the certificate. If you still can't delete the certificate, clear the internet browser cache and reopen the Azure portal in a new browser window.

Then try to delete the certificate.

## You can't purchase an App Service certificate

### Symptom

You can't purchase an [Azure App Service certificate](#) from the Azure portal.

### Cause and solution

This problem can occur for any of the following reasons:

- The App Service plan is Free or Shared. These pricing tiers don't support TLS.

**Solution:** Upgrade the App Service plan for app to Standard.

- The subscription doesn't have a valid credit card.

**Solution:** Add a valid credit card to your subscription.

- The subscription offer doesn't support purchasing an App Service certificate such as Microsoft Student.

**Solution:** Upgrade your subscription.

- The subscription reached the limit of purchases that are allowed on a subscription.

**Solution:** App Service certificates have a limit of 10 certificate purchases for the Pay-As-You-Go and EA subscription types. For other subscription types, the limit is 3. To increase the limit, contact [Azure support](#).

- The App Service certificate was marked as fraud. You received the following error message: "Your certificate has been flagged for possible fraud. The request is currently under review. If the certificate does not become usable within 24 hours, contact Azure Support."

**Solution:** If the certificate is marked as fraud and isn't resolved after 24 hours, follow these steps:

1. Sign in to the [Azure portal](#).
2. Go to **App Service Certificates**, and select the certificate.
3. Select **Certificate Configuration > Step 2: Verify > Domain Verification**. This step sends an email notice to the Azure certificate provider to resolve the problem.

## Custom domain problems

### A custom domain returns a 404 error

#### Symptom

When you browse to the site by using the custom domain name, you receive the following error message:

"Error 404-Web app not found."

#### Cause and solution

##### Cause 1

The custom domain that you configured is missing a CNAME or A record.

##### Solution for cause 1

- If you added an A record, make sure that a TXT record is also added. For more information, see [Create the A record](#).
- If you don't have to use the root domain for your app, we recommend that you use a CNAME record instead of an A record.
- Don't use both a CNAME record and an A record for the same domain. This issue can cause a conflict and prevent the domain from being resolved.

##### Cause 2

The internet browser might still be caching the old IP address for your domain.

## Solution for Cause 2

Clear the browser. For Windows devices, you can run the command `ipconfig /flushdns`. Use [WhatsmyDNS.net](#) to verify that your domain points to the app's IP address.

## You can't add a subdomain

### Symptom

You can't add a new host name to an app to assign a subdomain.

### Solution

- Check with subscription administrator to make sure that you have permissions to add a host name to the app.
- If you need more subdomains, we recommend that you change the domain hosting to Azure Domain Name Service (DNS). By using Azure DNS, you can add 500 host names to your app. For more information, see [Add a subdomain](#).

## DNS can't be resolved

### Symptom

You received the following error message:

"The DNS record could not be located."

### Cause

This problem occurs for one of the following reasons:

- The time to live (TTL) period has not expired. Check the DNS configuration for your domain to determine the TTL value, and then wait for the period to expire.
- The DNS configuration is incorrect.

### Solution

- Wait for 48 hours for this problem to resolve itself.
- If you can change the TTL setting in your DNS configuration, change the value to 5 minutes to see whether this resolves the problem.
- Use [WhatsmyDNS.net](#) to verify that your domain points to the app's IP address. If it doesn't, configure the A record to the correct IP address of the app.

## You need to restore a deleted domain

### Symptom

Your domain is no longer visible in the Azure portal.

### Cause

The owner of the subscription might have accidentally deleted the domain.

### Solution

If your domain was deleted fewer than seven days ago, the domain has not yet started the deletion process. In this case, you can buy the same domain again on the Azure portal under the same subscription. (Be sure to type the exact domain name in the search box.) You won't be charged again for this domain. If the domain was deleted more than seven days ago, contact [Azure support](#) for help with restoring the domain.

## Domain problems

### You purchased a TLS/SSL certificate for the wrong domain

#### Symptom

You purchased an App Service certificate for the wrong domain. You can't update the certificate to use the correct domain.

## **Solution**

Delete that certificate and then buy a new certificate.

If the current certificate that uses the wrong domain is in the "Issued" state, you'll also be billed for that certificate. App Service certificates are not refundable, but you can contact [Azure support](#) to see whether there are other options.

## **An App Service certificate was renewed, but the app shows the old certificate**

### **Symptom**

The App Service certificate was renewed, but the app that uses the App Service certificate is still using the old certificate. Also, you received a warning that the HTTPS protocol is required.

### **Cause**

App Service automatically syncs your certificate within 48 hours. When you rotate or update a certificate, sometimes the application is still retrieving the old certificate and not the newly updated certificate. The reason is that the job to sync the certificate resource hasn't run yet. Click Sync. The sync operation automatically updates the hostname bindings for the certificate in App Service without causing any downtime to your apps.

### **Solution**

You can force a sync of the certificate:

1. Sign in to the [Azure portal](#). Select **App Service Certificates**, and then select the certificate.
2. Select **Rekey and Sync**, and then select **Sync**. The sync takes some time to finish.
3. When the sync is completed, you see the following notification: "Successfully updated all the resources with the latest certificate."

## **Domain verification is not working**

### **Symptom**

The App Service certificate requires domain verification before the certificate is ready to use. When you select **Verify**, the process fails.

### **Solution**

Manually verify your domain by adding a TXT record:

1. Go to the Domain Name Service (DNS) provider that hosts your domain name.
2. Add a TXT record for your domain that uses the value of the domain token that's shown in the Azure portal.

Wait a few minutes for DNS propagation to run, and then select the **Refresh** button to trigger the verification.

As an alternative, you can use the HTML webpage method to manually verify your domain. This method allows the certificate authority to confirm the domain ownership of the domain that the certificate is issued for.

1. Create an HTML file that's named {domain verification token}.html. The content of this file should be the value of domain verification token.
2. Upload this file at the root of the web server that's hosting your domain.
3. Select **Refresh** to check the certificate status. It might take few minutes for verification to finish.

For example, if you're buying a standard certificate for `azure.com` with the domain verification token `1234abcd`, a web request made to <https://azure.com/1234abcd.html> should return `1234abcd`.

### **IMPORTANT**

A certificate order has only 15 days to complete the domain verification operation. After 15 days, the certificate authority denies the certificate, and you are not charged for the certificate. In this situation, delete this certificate and try again.

## **You can't purchase a domain**

### **Symptom**

You can't buy an App Service domain in the Azure portal.

#### Cause and solution

This problem occurs for one of the following reasons:

- There's no credit card on the Azure subscription, or the credit card is invalid.

**Solution:** Add a valid credit card to your subscription.

- You're not the subscription owner, so you don't have permission to purchase a domain.

**Solution:** Assign the Owner role to your account. Or contact the subscription administrator to get permission to purchase a domain.

- You have reached the limit for purchasing domains on your subscription. The current limit is 20.

**Solution:** To request an increase to the limit, contact [Azure support](#).

- Your Azure subscription type does not support the purchase of an App Service domain.

**Solution:** Upgrade your Azure subscription to another subscription type, such as a Pay-As-You-Go subscription.

#### You can't add a host name to an app

##### Symptom

When you add a host name, the process fails to validate and verify the domain.

##### Cause

This problem occurs for one of the following reasons:

- You don't have permission to add a host name.

**Solution:** Ask the subscription administrator to give you permission to add a host name.

- Your domain ownership could not be verified.

**Solution:** Verify that your CNAME or A record is configured correctly. To map a custom domain to an app, create either a CNAME record or an A record. If you want to use a root domain, you must use A and TXT records:

RECORD TYPE	HOST	POINT TO
A	@	IP address for an app
TXT	@	<app-name>.azurewebsites.net
CNAME	www	<app-name>.azurewebsites.net

## FAQ

### Do I have to configure my custom domain for my website once I buy it?

When you purchase a domain from the Azure portal, the App Service application is automatically configured to use that custom domain. You don't have to take any additional steps. For more information, watch [Azure App Service Self Help: Add a Custom Domain Name](#) on Channel9.

### Can I use a domain purchased in the Azure portal to point to an Azure VM instead?

Yes, you can point the domain to a VM. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

## **Is my domain hosted by GoDaddy or Azure DNS?**

App Service Domains use GoDaddy for domain registration and Azure DNS to host the domains.

## **I have auto-renew enabled but still received a renewal notice for my domain via email. What should I do?**

If you have auto-renew enabled, you do not need to take any action. The notice email is provided to inform you that the domain is close to expiring and to renew manually if auto-renew is not enabled.

## **Will I be charged for Azure DNS hosting my domain?**

The initial cost of domain purchase applies to domain registration only. In addition to the registration cost, there are incurring charges for Azure DNS based on your usage. For more information, see [Azure DNS pricing](#) for more details.

## **I purchased my domain earlier from the Azure portal and want to move from GoDaddy hosting to Azure DNS hosting. How can I do this?**

It is not mandatory to migrate to Azure DNS hosting. If you do want to migrate to Azure DNS, the domain management experience in the Azure portal about provides information on steps necessary to move to Azure DNS. If the domain was purchased through App Service, migration from GoDaddy hosting to Azure DNS is relatively seamless procedure.

## **I would like to purchase my domain from App Service Domain but can I host my domain on GoDaddy instead of Azure DNS?**

Beginning July 24, 2017, App Service domains purchased in the portal are hosted on Azure DNS. If you prefer to use a different hosting provider, you must go to their website to obtain a domain hosting solution.

## **Do I have to pay for privacy protection for my domain?**

When you purchase a domain through the Azure portal, you can choose to add privacy at no additional cost. This is one of the benefits of purchasing your domain through Azure App Service.

## **If I decide I no longer want my domain, can I get my money back?**

When you purchase a domain, you are not charged for a period of five days, during which time you can decide that you do not want the domain. If you do decide you don't want the domain within that five-day period, you are not charged. (.uk domains are an exception to this. If you purchase a .uk domain, you are charged immediately and you cannot be refunded.)

## **Can I use the domain in another Azure App Service app in my subscription?**

Yes. When you access the Custom Domains and TLS blade in the Azure portal, you see the domains that you have purchased. You can configure your app to use any of those domains.

## **Can I transfer a domain from one subscription to another subscription?**

You can move a domain to another subscription/resource group using the [Move-AzResource](#) PowerShell cmdlet.

## **How can I manage my custom domain if I don't currently have an Azure App Service app?**

You can manage your domain even if you don't have an App Service Web App. Domain can be used for Azure services like Virtual machine, Storage etc. If you intend to use the domain for App Service Web Apps, then you need to include a Web App that is not on the Free App Service plan in order to bind the domain to your web app.

## **Can I move a web app with a custom domain to another subscription or from App Service Environment v1 to V2?**

Yes, you can move your web app across subscriptions. Follow the guidance in [How to move resources in Azure](#). There are a few limitations when moving the web app. For more information, see [Limitations for moving App Service resources](#).

After moving the web app, the host name bindings of the domains within the custom domains setting should remain the same. No additional steps are required to configure the host name bindings.

# How to prepare for an inbound IP address change

11/2/2021 • 2 minutes to read • [Edit Online](#)

If you received a notification that the inbound IP address of your Azure App Service app is changing, follow the instructions in this article.

## Determine if you have to do anything

- Option 1: If your App Service app does not have a Custom Domain, no action is required.
- Option 2: If only a CNAME record (DNS record pointing to a URI) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), no action is required.
- Option 3: If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the new one. You can find the new IP address by following the instructions in the next section.
- Option 4: If your application is behind a load balancer, IP Filter, or any other IP mechanism that requires your app's IP address, replace the existing IP address with the new one. You can find the new IP address by following the instructions in the next section.

## Find the new inbound IP Address in the Azure portal

The new inbound IP address that is being given to your app is in the portal in the **Virtual IP address** field. Both this new IP address and the old one are connected to your app now, and later the old one will be disconnected.

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. If the app is a function app, see [Function app inbound IP address](#).
5. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.
6. Copy the IP address and reconfigure your domain record or IP mechanism.

## Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

# How to prepare for an outbound IP address change

11/2/2021 • 2 minutes to read • [Edit Online](#)

If you received a notification that the outbound IP addresses of your Azure App Service app are changing, follow the instructions in this article.

## Determine if you have to do anything

- Option 1: If your App Service app does not use IP filtering, an explicit inclusion list, or special handling of outbound traffic such as routing or firewall, no action is required.
- Option 2: If your app does have special handling for the outbound IP addresses (see examples below), add the new outbound IP addresses wherever the existing ones appear. Don't replace the existing IP addresses. You can find the new outbound IP addresses by following the instructions in the next section.

For example, an outbound IP address may be explicitly included in a firewall outside your app, or an external payment service may have an allowed list that contains the outbound IP address for your app. If your outbound address is configured in a list anywhere outside your app, that needs to change.

## Find the outbound IP addresses in the Azure portal

The new outbound IP addresses are shown in the portal before they take effect. When Azure starts using the new ones, the old ones will no longer be used. Only one set at a time is used, so entries in inclusion lists must have both old and new IP addresses to prevent an outage when the switch happens.

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. If the app is a function app, see [Function app outbound IP addresses](#).
5. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Outbound IP addresses**.
6. Copy the IP addresses, and add them to your special handling of outbound traffic such as a filter or allowed list. Don't delete the existing IP addresses in the list.

## Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).

# How to prepare for a TLS/SSL IP address change

11/2/2021 • 2 minutes to read • [Edit Online](#)

If you received a notification that the TLS/SSL IP address of your Azure App Service app is changing, follow the instructions in this article to release existing TLS/SSL IP address and assign a new one.

## NOTE

Service Endpoint is not currently supported when enabling IP Based SSL on App Service TLS/SSL bindings.

## Release TLS/SSL IP addresses and assign new ones

1. Open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **SSL settings** in the left navigation.
5. In the TLS/SSL bindings section, select the host name record. In the editor that opens, choose **SNI SSL** on the **SSL Type** drop-down menu and click **Add Binding**. When you see the operation success message, the existing IP address has been released.
6. In the **SSL bindings** section, again select the same host name record with the certificate. In the editor that opens, this time choose **IP Based SSL** on the **SSL Type** drop-down menu and click **Add Binding**. When you see the operation success message, you've acquired a new IP address.
7. If an A record (DNS record pointing directly to your IP address) is configured in your Domain Registration Portal (third party DNS Provider or Azure DNS), replace the existing IP address with the newly generated one. You can find the new IP address by following the instructions in the next section.

## Find the new SSL IP address in the Azure Portal

1. Wait a few minutes, and then open the [Azure portal](#).
2. In the left-hand navigation menu, select **App Services**.
3. Select your App Service app from the list.
4. Under the **Settings** header, click **Properties** in the left navigation, and find the section labeled **Virtual IP address**.
5. Copy the IP address and reconfigure your domain record or IP mechanism.

## Next steps

This article explained how to prepare for an IP address change that was initiated by Azure. For more information about IP addresses in Azure App Service, see [Inbound and outbound IP addresses in Azure App Service](#).