

# Contents

## [App Service on Linux Documentation](#)

### [Overview](#)

[About Web Apps](#)

[About App Service on Linux](#)

[Compare hosting options](#)

### [Quickstarts](#)

[Create .NET Core app](#)

[Create PHP app](#)

[Create Node.js app](#)

[Create Java app](#)

[Create Python app](#)

[Create Ruby app](#)

[Run custom container](#)

[Create multi-container app](#)

### [Tutorials](#)

[App with DB](#)

[.NET Core with SQL DB](#)

[Ruby with Postgres](#)

[PHP with MySQL](#)

[Node.js with MongoDB](#)

[Python with Postgres](#)

[Java with Spring Data](#)

[Multi-container app](#)

[Run container from ACR](#)

[Map custom domain](#)

[Create SSL binding for domain](#)

[Add CDN](#)

[Authenticate users](#)

### [Samples](#)

[Azure CLI](#)

[Azure PowerShell](#)

## Concepts

[App Service plans](#)

[App Service Environment](#)

[Inbound and outbound IPs](#)

[Authentication and authorization](#)

[Choose deployment type](#)

[Diagnostics](#)

## How-To guides

[Configure app](#)

[Configure common settings](#)

[Configure ASP.NET Core](#)

[Configure Node.js](#)

[Configure PHP](#)

[Configure Java](#)

[Configure Python](#)

[Configure Ruby](#)

[Configure custom container](#)

[Serve content with Storage](#)

## Deploy to Azure

[Deploy the app](#)

[Run from package](#)

[Deploy via FTP](#)

[Deploy via cloud sync](#)

[Deploy continuously](#)

[Deploy from local Git](#)

[Deploy with GitHub Actions](#)

[Deploy with template](#)

[Java SE with Maven](#)

[Set deployment credentials](#)

[Create staging environments](#)

[Map custom domain](#)

[Buy domain](#)

[Map domains with Traffic Manager](#)

[Migrate an active domain](#)

[Secure app](#)

[Add SSL cert](#)

[Authenticate users](#)

[Authenticate with Azure AD](#)

[Authenticate with Facebook](#)

[Authenticate with Google](#)

[Authenticate with Microsoft account](#)

[Authenticate with Twitter](#)

[Advanced auth](#)

[Configure TLS mutual authentication](#)

[Scale app](#)

[Scale up server capacity](#)

[Scale out to multiple instances](#)

[High density hosting](#)

[Monitor app](#)

[Connect to app via SSH](#)

[Quotas & alerts](#)

[Enable logs](#)

[Manage app](#)

[Back up an app](#)

[Restore a backup](#)

[Clone an app](#)

[Move resources](#)

[Reference](#)

[Azure CLI](#)

[Azure PowerShell](#)

[REST API](#)

[Resources](#)

[App Service on Linux FAQ](#)

[Azure Roadmap](#)

[Pricing](#)

[Quota Information](#)

[Service Updates](#)

[Migration Assistant guides](#)

[Best practices](#)

[Samples](#)

[Videos](#)

[Cookbooks](#)

[Reference Architectures](#)

[Deployment Scripts](#)

[Troubleshooting](#)

[Troubleshoot with Visual Studio](#)

[Troubleshoot Node.js app](#)

[Troubleshoot HTTP 502 & 503](#)

[Troubleshoot performance issues](#)

[FAQ](#)

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

[Deployment FAQ](#)

[Open source technologies FAQ](#)

[Configuration and management FAQ](#)

# App Service overview

12/2/2019 • 2 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. For Linux-based environments, see [App Service on Linux](#).

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 SSL certificates.

With App Service, you pay for the Azure compute resources you use. The compute resources you use is 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.
- **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](#) or with social login ([Google](#), [Facebook](#), [Twitter](#), and [Microsoft](#)). 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 integration** - Dedicated tools in Visual Studio 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 [Service Fabric](#). If you need more control over the VMs that your code runs on, 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.](#)

## Next steps

Create your first web app.

[ASP.NET Core](#)

[ASP.NET](#)

[PHP](#)

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

[Node.js](#)

[Java](#)

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

[HTML](#)

# Introduction to Azure App Service on Linux

12/2/2019 • 2 minutes to read • [Edit Online](#)

[Azure App Service](#) is a fully managed compute platform that is optimized for hosting websites and web applications. Customers can use App Service on Linux to host web apps natively on Linux for supported application stacks.

## Languages

App Service on Linux supports a number of Built-in images in order to increase developer productivity.

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, there are instructions on how to [build your own Docker image](#) to deploy to Web App for Containers.

## Deployments

- FTP
- Local Git
- GitHub
- Bitbucket

## DevOps

- Staging environments
- [Azure Container Registry](#) and DockerHub CI/CD

## Console, Publishing, and Debugging

- Environments
- Deployments
- Basic console
- SSH

## Scaling

- Customers can scale web apps up and down by changing the tier of their [App Service plan](#)

## Locations

Check the [Azure Status Dashboard](#).

## Limitations

The Azure portal shows only features that currently work for Web App for Containers. As we enable more features, they will become visible on the portal.

App Service on Linux is only supported with [Free, Basic, Standard, and Premium](#) app service plans and does not have a [Shared](#) tier. You cannot create a Linux Web App in an App Service plan already hosting non-Linux

Web Apps.

Based on a current limitation, for the same resource group you cannot mix Windows and Linux apps in the same region.

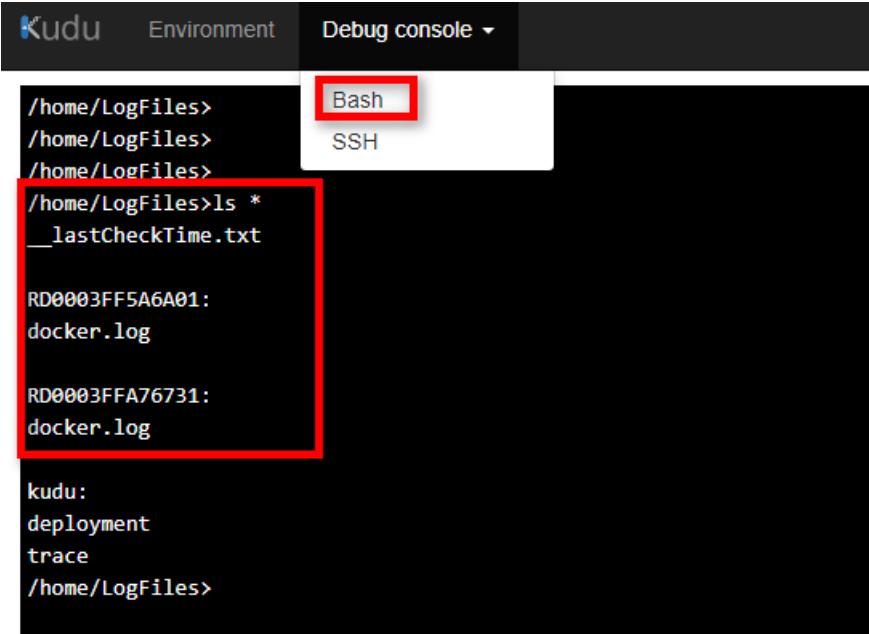
## Troubleshooting

### NOTE

There's new integrated logging capability with [Azure Monitoring \(preview\)](#).

When your application fails to start or you want to check the logging from your app, check the Docker logs in the LogFiles directory. You can access this directory either through your SCM site or via FTP. To log the `stdout` and `stderr` from your container, you need to enable **Application Logging** under **App Service Logs**. The setting takes effect immediately. App Service detects the change and restarts the container automatically.

You can access the SCM site from **Advanced Tools** in the **Development Tools** menu.



```
/home/LogFiles>
/home/LogFiles>
/home/LogFiles>
/home/LogFiles>ls *
__lastCheckTime.txt

RD0003FF5A6A01:
docker.log

RD0003FFA76731:
docker.log

kudu:
deployment
trace
/home/LogFiles>
```

## Next steps

The following articles get you started with App Service on Linux with web apps written in a variety of languages:

- [.NET Core](#)
- [PHP](#)
- [Node.js](#)
- [Java](#)
- [Python](#)
- [Ruby](#)
- [Go](#)
- [Multi-container apps](#)

For more information on App Service on Linux, see:

- [App Service for Linux FAQ](#)
- [SSH support for App Service on Linux](#)

- Set up staging environments in App Service
- Docker Hub continuous deployment

You can post questions and concerns on [our forum](#).

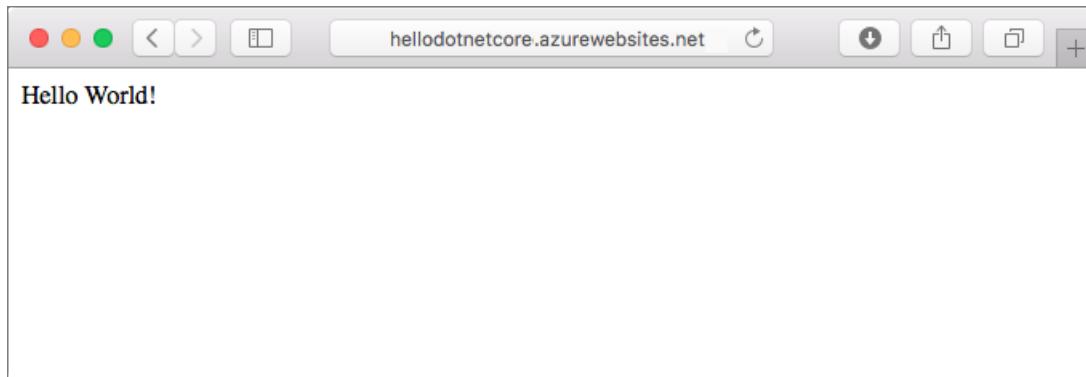
# Create an ASP.NET Core app in App Service on Linux

2/21/2020 • 7 minutes to read • [Edit Online](#)

## NOTE

This article deploys an app to App Service on Linux. To deploy to App Service on *Windows*, see [Create an ASP.NET Core app in Azure](#).

[App Service on Linux](#) provides a highly scalable, self-patching web hosting service using the Linux operating system. This quickstart shows how to create a [.NET Core](#) app on App Service on Linux. You create the app using the [Azure CLI](#), and you use Git to deploy the .NET Core code to the app.



You can follow the steps in this article using a Mac, Windows, or Linux machine.

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

## Prerequisites

To complete this quickstart:

- [Install Git](#)
- [Install .NET Core](#)

## Create the app locally

In a terminal window on your machine, create a directory named `hellodotnetcore` and change the current directory to it.

```
mkdir hellodotnetcore
cd hellodotnetcore
```

Create a new .NET Core app.

```
dotnet new web
```

## Run the app locally

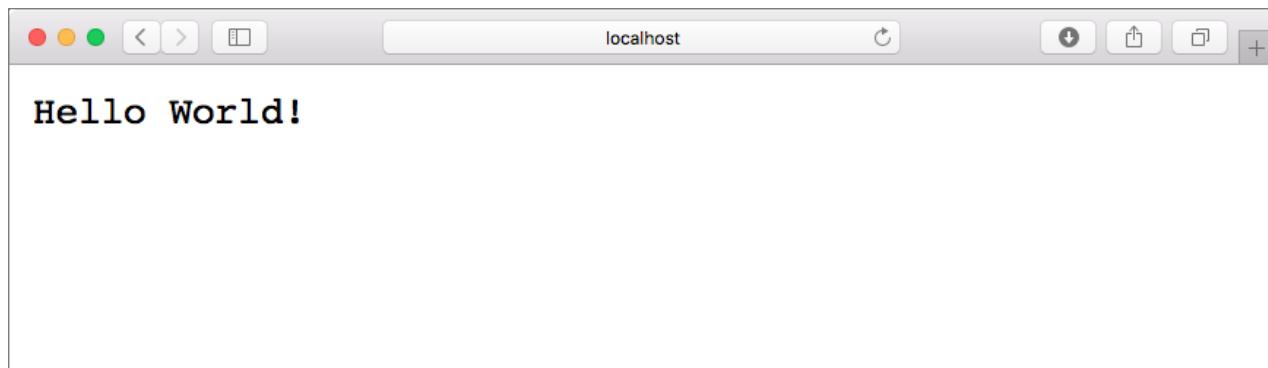
Run the application locally so that you see how it should look when you deploy it to Azure.

Restore the NuGet packages and run the app.

```
dotnet run
```

Open a web browser, and navigate to the app at <http://localhost:5000>.

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



In your terminal window, press **Ctrl+C** to exit the web server. Initialize a Git repository for the .NET Core project.

```
git init  
git add .  
git commit -m "first commit"
```

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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" with a white play icon. A red box highlights this button.
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	A screenshot of the Azure portal's top navigation bar. The "Cloud Shell" button, which is a blue square with a white terminal icon, is highlighted with a red box.
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	A screenshot of the Azure portal's top navigation bar. The "Cloud Shell" button, which is a blue square with a white terminal 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 like 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 in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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 `DOTNETCORE|2.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash  
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime  
"DOTNETCORE|2.2" --deployment-local-git  
# PowerShell  
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime  
"DOTNETCORE|2.2" --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 web app in a Linux container, with git deployment enabled.

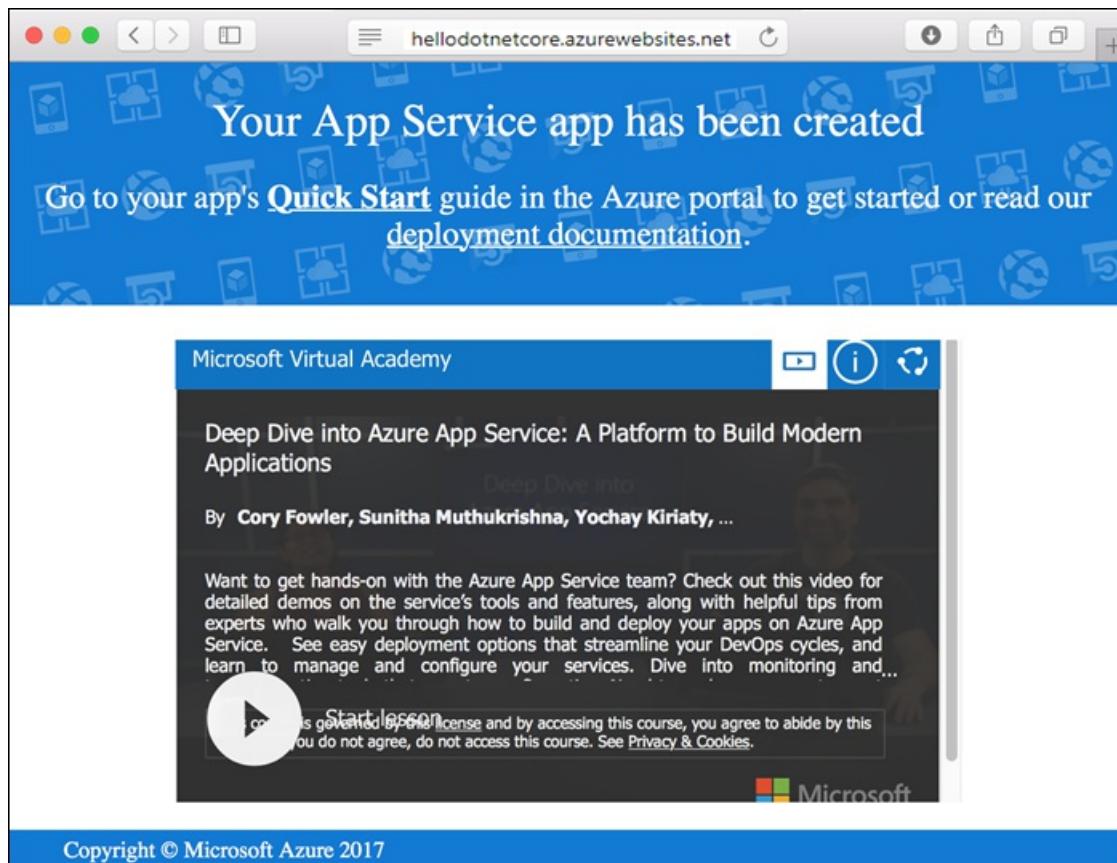
### NOTE

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.

Browse to your newly created app. Replace `<app-name>` with your app name.

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

Here is what your new app should look like:



## Push to Azure from Git

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>
```

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 master
```

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

```
Counting objects: 22, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (18/18), done.  
Writing objects: 100% (22/22), 51.21 KiB | 3.94 MiB/s, done.  
Total 22 (delta 1), reused 0 (delta 0)  
remote: Updating branch 'master'.  
remote: Updating submodules.  
remote: Preparing deployment for commit id '741f16d1db'.  
remote: Generating deployment script.  
remote: Project file path: ./hellodotnetcore.csproj  
remote: Generated deployment script files  
remote: Running deployment command...  
remote: Handling ASP.NET Core Web Application deployment.  
remote: .....  
remote: Restoring packages for /home/site/repository/hellodotnetcore.csproj...  
remote: .....  
remote: Installing System.Xml.XPath 4.0.1.  
remote: Installing System.Diagnostics.Tracing 4.1.0.  
remote: Installing System.Threading.Tasks.Extensions 4.0.0.  
remote: Installing System.Reflection.Emit.ILGeneration 4.0.1.  
remote: ...  
remote: Finished successfully.  
remote: Running post deployment command(s)...  
remote: Deployment successful.  
To https://<app-name>.scm.azurewebsites.net/<app-name>.git  
 * [new branch]      master -> master
```

## Browse to the app

Browse to the deployed application using your web browser.

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

The .NET Core sample code is running in App Service on Linux with a built-in image.



**Congratulations!** You've deployed your first .NET Core app to App Service on Linux.

## Update and redeploy the code

In the local directory, open the *Startup.cs* file. Make a small change to the text in the method call

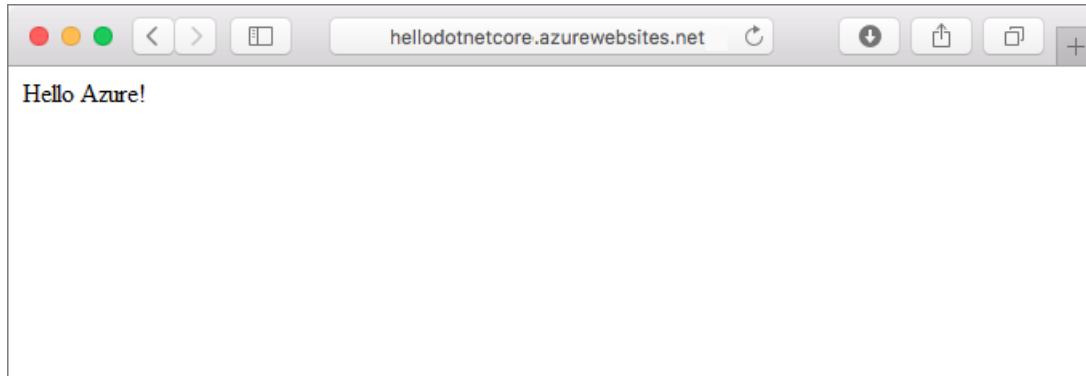
```
context.Response.WriteAsync :
```

```
await context.Response.WriteAsync("Hello Azure!");
```

Commit your changes in Git, and then push the code changes to Azure.

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

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



## Manage your new Azure app

Go to the [Azure portal](#) to manage the app you created.

From the left menu, click **App Services**, and then click the name of your Azure app.

A screenshot of the Microsoft Azure portal's App Services overview page. The top navigation bar shows "Microsoft Azure" and "App Services". The main area is titled "App Services" under "cephaslinhotmail (Default Directory)". A sidebar on the left lists "App Services", "Web", "API", "Function", "Container", "Mobile", and "Data". The main content area shows "Subscriptions: All 2 selected" with filters for "Filter by name...", "All subscriptions", "All resource gr...", "All locations", and "No grouping". A table lists "1 items":

NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCRIPTION	...
helloworldnetcore	Running	Web app	myAppServicePlan	West Europe	Visual Studio Ultimate wi...	...

The "helloworldnetcore" row is highlighted with a red box.

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

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:

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

This command may take a minute to run.

## Next steps

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

[Configure ASP.NET Core app](#)

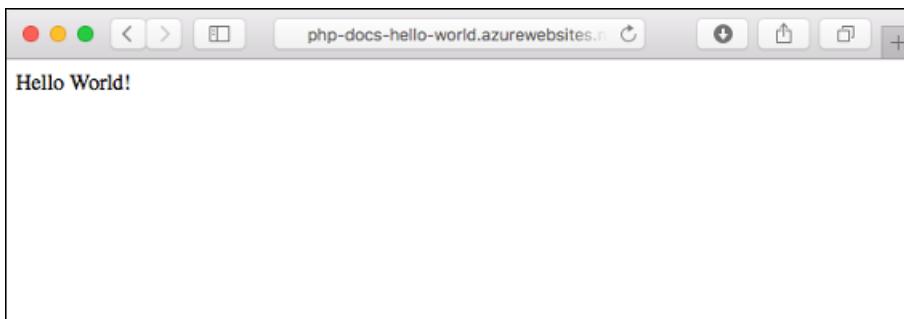
# Create a PHP app in App Service on Linux

2/21/2020 • 7 minutes to read • [Edit Online](#)

## NOTE

This article deploys an app to App Service on Linux. To deploy to App Service on Windows, see [Create a PHP app in 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 PHP app to Azure App Service on Linux using the [Cloud Shell](#).



You can follow the steps in this article using a Mac, Windows, or Linux machine.

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

In a terminal window, run the following commands to 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
```

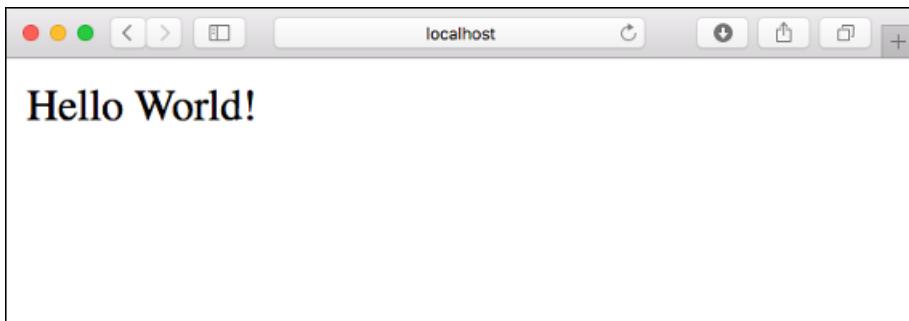
## Run the app locally

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
```

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.



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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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.

## 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 like 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 in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "West Europe",
  "hostingEnvironmentProfile": null,
  "id": "/subscriptions/0000-0000-0000-0000/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|7.0`. 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|7.0" --deployment-local-git
# PowerShell
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.0" --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 new web app, with git deployment enabled.

#### NOTE

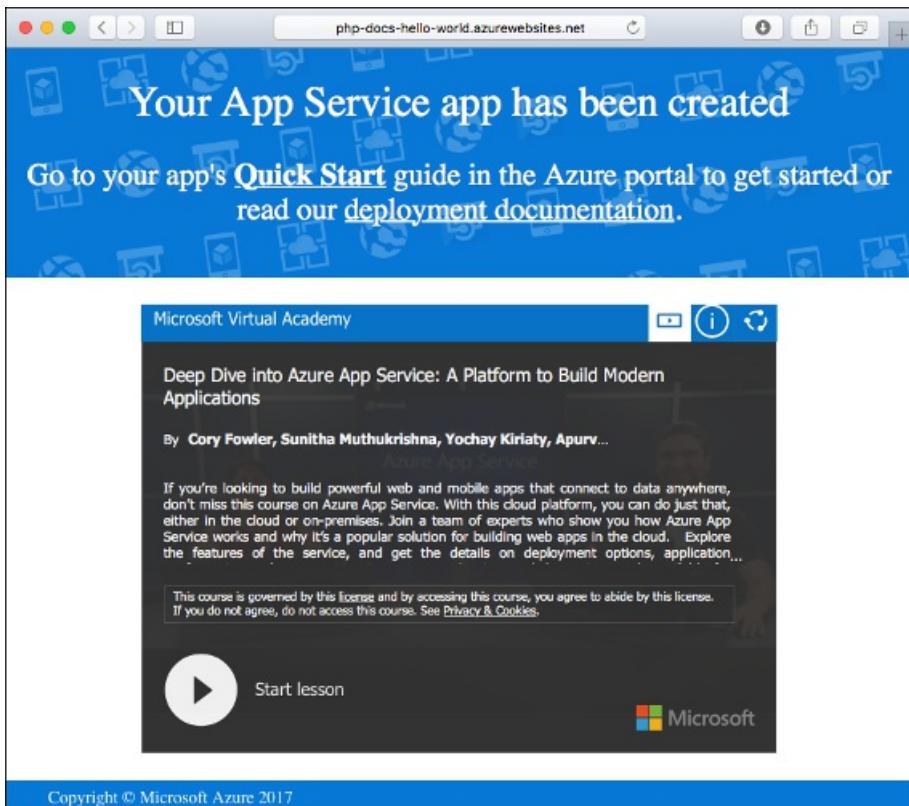
The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

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

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

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

Here is what your new app should look like:



## Push to Azure from Git

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>
```

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 master
```

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

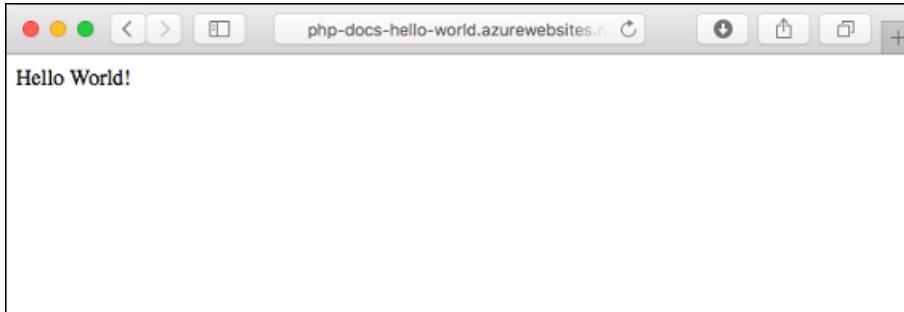
```
Counting objects: 2, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (2/2), 352 bytes | 0 bytes/s, done.  
Total 2 (delta 1), reused 0 (delta 0)  
remote: Updating branch 'master'.  
remote: Updating submodules.  
remote: Preparing 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 master -> master
```

## 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 App Service on Linux with built-in image.



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

## Update locally and redeploy the code

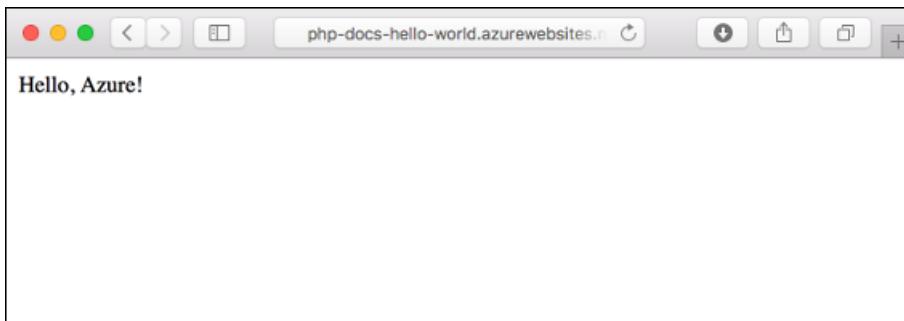
In the local directory, open the `index.php` file within the PHP app, and make a small change to the text within the string next to `echo`:

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

Commit your changes in Git, and then push the code changes to Azure.

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

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



## Manage your new Azure app

Go to the [Azure portal](#) to manage the app you created.

From the left menu, click **App Services**, and then click the name of your Azure app.

NAME	RESOURCE GROUP	LOCATION	STATUS	APP SERVICE PLAN
php-docs-hello-world	myResourceGroup	West Europe	Running	

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

Overview

Resource group: myResourceGroup  
Status: Running  
Location: West Europe  
Subscription name: Field Engineer Demo  
Subscription ID: 5d6c94cd-6781-43e3-8a94-ceef4c28850e

URL: http://php-docs-hello-world.azurewebsites.net  
App Service plan/pricing tier: quickStartPlan (Standard: 1 Small)  
Git Deployment username: demoaccount  
Git clone url: https://demoaccount@php-docs-hello-wor...  
FTP hostname: ftp://www-prod-am2-085.ftp.azurewebsite...

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:

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

This command may take a minute to run.

## Next steps

[Tutorial: PHP app with MySQL](#)

[Configure PHP app](#)

# Create a Node.js app in Azure

12/2/2019 • 5 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This quickstart shows how to deploy a Node.js app to Azure App Service.

## Prerequisites

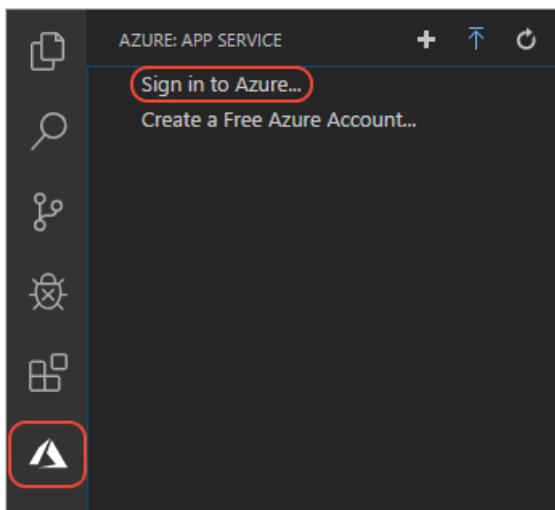
If you don't have an Azure account, [sign up today](#) for a free account with \$200 in Azure credits to try out any combination of services.

You need [Visual Studio Code](#) installed along with [Nodejs](#) and [npm](#), the Node.js package manager.

You will also need to install the [Azure App Service extension](#), which you can use to create, manage, and deploy Linux Web Apps on the Azure Platform as a Service (PaaS).

### Sign in

Once the extension is installed, log into your Azure account. In the Activity Bar, select the Azure logo to show the **AZURE APP SERVICE** explorer. Select **Sign in to Azure...** and follow the instructions.



### Troubleshooting

If you see the error "**Cannot find subscription with name [subscription ID]**", it might be because you're behind a proxy and unable to reach the Azure API. Configure `HTTP_PROXY` and `HTTPS_PROXY` environment variables with your proxy information in your terminal using `export`.

```
export HTTPS_PROXY=https://username:password@proxy:8080
export HTTP_PROXY=http://username:password@proxy:8080
```

If setting the environment variables doesn't correct the issue, contact us by selecting the **I ran into an issue** button below.

### Prerequisite check

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

In VS 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

## Create your Node.js application

Next, create a Node.js application that can be deployed to the Cloud. This quickstart uses an application generator to quickly scaffold out the application from a terminal.

### TIP

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

### Scaffold a new application with the Express Generator

[Express](#) is a popular framework for building and running Node.js applications. You can scaffold (create) a new Express application using the [Express Generator](#) tool. The Express Generator is shipped as an npm module and can be run directly (without installation) by using the npm command-line tool `npx`.

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

The `--view pug --git` parameters tell the generator to use the [pug](#) template engine (formerly known as [jade](#)) and to create a `.gitignore` file.

To install all of the application's dependencies, go to the new folder and run `npm install`.

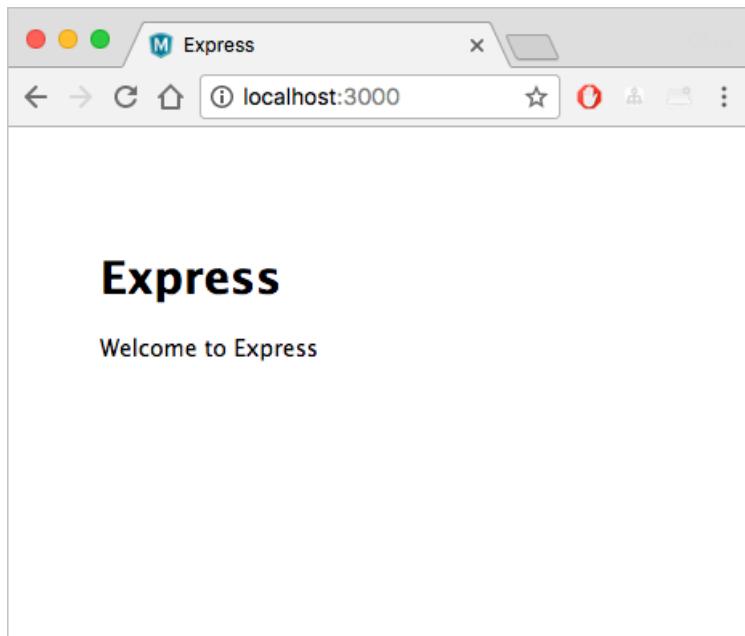
```
cd myExpressApp  
npm install
```

### Run the application

Next, ensure that the application runs. From the terminal, start the application using the `npm start` command to start the server.

```
npm start
```

Now, open your browser and navigate to <http://localhost:3000>, where you should see something like this:



I ran into an issue

# Deploy to Azure

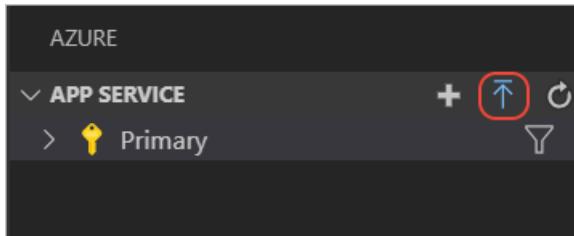
In this section, you deploy your Node.js app using VS Code and the Azure App Service extension. This quickstart uses the most basic deployment model where your app is zipped and deployed to an Azure Web App on Linux.

## Deploy using Azure App Service

First, open your application folder in VS Code.

```
code .
```

In the **AZURE APP SERVICE** explorer, select the blue up arrow icon to deploy your app to Azure.



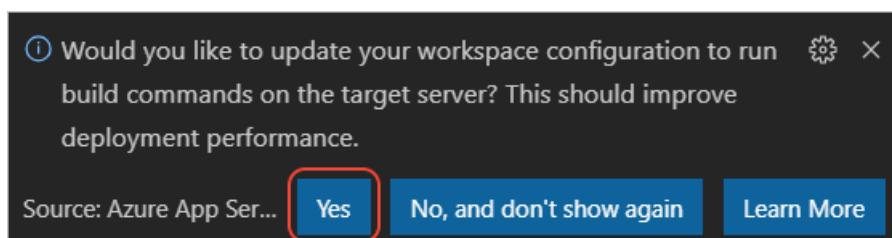
### TIP

You can also deploy from the **Command Palette** (CTRL + SHIFT + P) by typing 'deploy to web app' and running the **Azure App Service: Deploy to Web App** command.

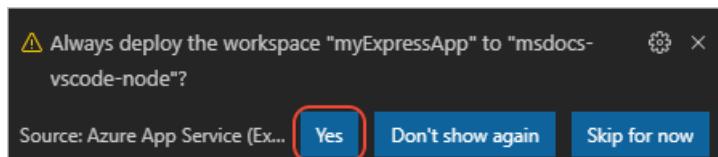
1. Choose the directory that you currently have open, `myExpressApp`.
2. Choose **Create new Web App**, which deploys to App Service on Linux by default.
3. Type a globally unique name for your Web App and press ENTER. Valid characters for an app name are 'a-z', '0-9', and '-'.
4. Choose your **Node.js version**, LTS is recommended.

The notification channel shows the Azure resources that are being created for your app.

5. Select **Yes** when prompted to update your configuration to run `npm install` on the target server. Your app is then deployed.



6. When the deployment starts, you're prompted to update your workspace so that later deployments will automatically target the same App Service Web App. Choose **Yes** to ensure your changes are deployed to the correct app.



## TIP

Be sure that your application is listening on the port provided by the PORT environment variable: `process.env.PORT`.

## Browse the app in Azure

Once the deployment completes, select **Browse Website** in the prompt to view your freshly deployed web app.

## Troubleshooting

If you see the error "**You do not have permission to view this directory or page.**", then the application probably failed to start correctly. Head to the next section and view the log output to find and fix the error. If you aren't able to fix it, contact us by selecting the **I ran into an issue** button below. We're happy to help!

[I ran into an issue](#)

## Update the app

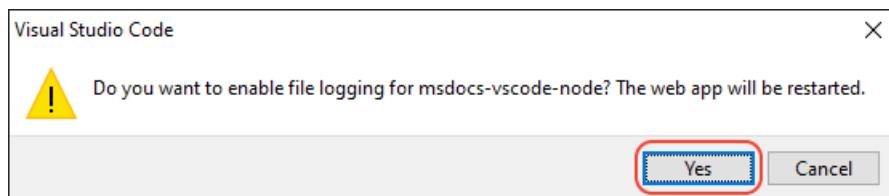
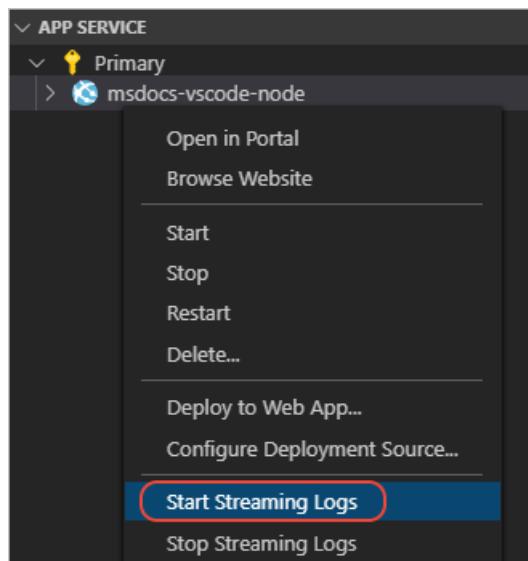
You can deploy changes to this app by using the same process and choosing the existing app rather than creating a new one.

## Viewing Logs

In this section, you learn how to view (or "tail") the logs from the running App Service app. Any calls to `console.log` in the app are displayed in the output window in Visual Studio Code.

Find the app in the **AZURE APP SERVICE** explorer, right-click the app, and choose **View Streaming Logs**.

When prompted, choose to enable logging and restart the application. Once the app is restarted, the VS Code output window opens with a connection to the log stream.



After a few seconds, you'll see a message indicating that you're connected to the log-streaming service. Refresh the page a few times to see more activity.

```
```bash
2019-09-20 20:37:39.574 INFO  - Initiating warmup request to container msdocs-vscode-node_2_00ac292a for site
msdocs-vscode-node
2019-09-20 20:37:55.011 INFO  - Waiting for response to warmup request for container msdocs-vscode-
node_2_00ac292a. Elapsed time = 15.4373071 sec
2019-09-20 20:38:08.233 INFO  - Container msdocs-vscode-node_2_00ac292a for site msdocs-vscode-node
initialized successfully and is ready to serve requests.
2019-09-20T20:38:21  Startup Request, url: /Default.cshtml, method: GET, type: request, pid: 61,1,7,
SCM_SKIP_SSL_VALIDATION: 0, SCM_BIN_PATH: /opt/Kudu/bin, ScmType: None
```

```

I ran into an issue

## Next steps

Congratulations, you've successfully completed this quickstart!

Next, 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.

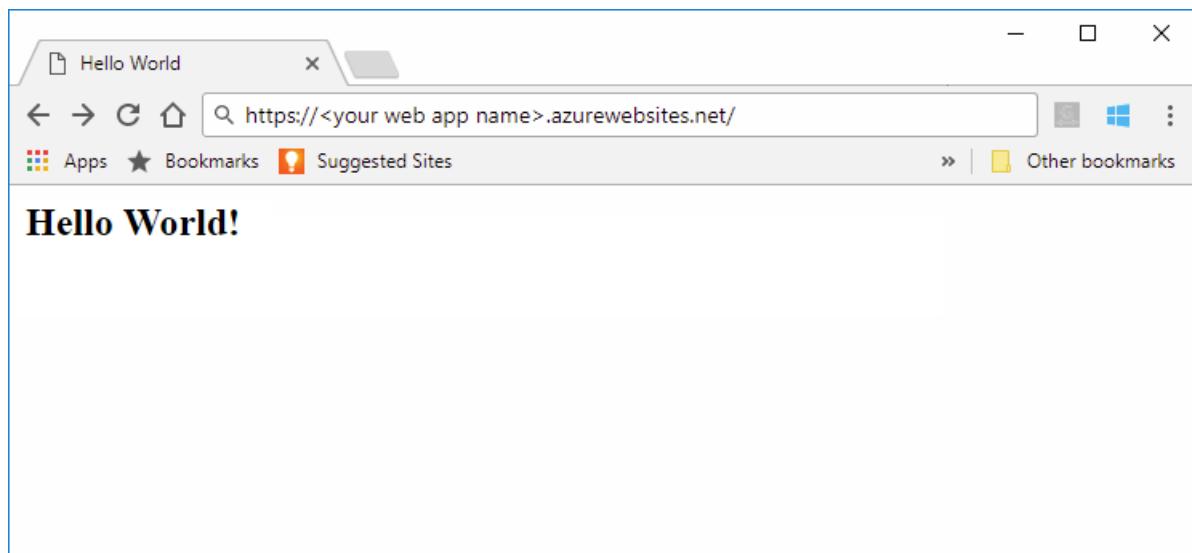
# Quickstart: Create a Java app on Azure App Service on Linux

2/18/2020 • 4 minutes to read • [Edit Online](#)

App Service on Linux provides a highly scalable, self-patching, web hosting service using the Linux operating system. This quickstart shows how to use the [Azure CLI](#) with the [Azure Web App Plugin for Maven](#) to deploy a Java web archive (WAR) file on the Linux operating system.

## NOTE

The same thing can also be done using popular IDEs like IntelliJ, Eclipse and VS Code. Check out our similar documents at [Azure Toolkit for IntelliJ Quickstart](#), [Azure Toolkit for Eclipse Quickstart](#) or [VS Code Quickstart](#).



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 <b>Try It</b> in the upper-right corner of a code block.<br>Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.            | A screenshot of a code editor showing a line of code. To its right is a 'Copy' button and a 'Try It' button, which is highlighted with a red box. |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. | A screenshot of the Azure portal's top navigation bar. The 'Cloud Shell' button is highlighted with a red box.                                    |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .   | A screenshot of the Azure portal's top navigation bar. The 'Cloud Shell' button 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.

## Create a Java app

Execute the following Maven command in the Cloud Shell prompt to create a new app named `helloworld`:

```
mvn archetype:generate "-DgroupId=example.demo" "-DartifactId=helloworld" "-DarchetypeArtifactId=maven-archetype-webapp"
```

Then change your working directory to the project folder:

```
cd helloworld
```

## Configure the Maven plugin

The deploy process to Azure App Service uses account credentials from the Azure CLI. [Sign in with the Azure CLI](#) before continuing.

```
az login
```

Then you can configure the deployment, run the maven command in the Command Prompt and use the default configurations by pressing **ENTER** until you get the **Confirm (Y/N)** prompt, then press '**y**' and the configuration is done.

```
mvn com.microsoft.azure:azure-webapp-maven-plugin:1.8.0:config
```

A sample process looks like:

```

~@Azure:~/helloworld$ mvn com.microsoft.azure:azure-webapp-maven-plugin:1.8.0:config
[INFO] Scanning for projects...
[INFO]
[INFO] -----< example.demo:helloworld >-----
[INFO] Building helloworld Maven Webapp 1.0-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] --- azure-webapp-maven-plugin:1.8.0:config (default-cli) @ helloworld ---
[WARNING] The plugin may not work if you change the os of an existing webapp.
Define value for OS(Default: Linux):
1. linux [*]
2. windows
3. docker
Enter index to use:
Define value for javaVersion(Default: jre8):
1. Java 11
2. Java 8 [*]
Enter index to use:
Define value for runtimeStack(Default: TOMCAT 8.5):
1. TOMCAT 9.0
2. TOMCAT 8.5 [*]
Enter index to use:
Please confirm webapp properties
AppName : helloworld-1558400876966
ResourceGroup : helloworld-1558400876966-rg
Region : westeurope
PricingTier : Premium_P1V2
OS : Linux
RuntimeStack : TOMCAT 8.5-jre8
Deploy to slot : false
Confirm (Y/N)? : Y

```

#### NOTE

In this article we are only working with Java apps packaged in WAR files. The plugin also supports JAR web applications, visit [Deploy a Java SE JAR file to App Service on Linux](#) to try it out.

Navigate to `pom.xml` again to see the plugin configuration is updated, You can modify other configurations for App Service directly in your pom file if needed, some common ones are listed below:

| PROPERTY                           | REQUIRED | DESCRIPTION  | VERSION |
|------------------------------------|----------|--|---------|
| <code>&lt;schemaVersion&gt;</code> | false    | Specify the version of the configuration schema.<br>Supported values are: <code>v1</code> , <code>v2</code> .  | 1.5.2   |
| <code>&lt;resourceGroup&gt;</code> | true     | Azure Resource Group for your Web App.   | 0.1.0+  |
| <code>&lt;appName&gt;</code>       | true     | The name of your Web App.  | 0.1.0+  |
| <code>&lt;region&gt;</code>        | true     | Specifies the region where your Web App will be hosted; the default value is <b>westeurope</b> . All valid regions at <a href="#">Supported Regions</a> section. | 0.1.0+  |

| PROPERTY      | REQUIRED | DESCRIPTION  | VERSION |
|---------------|----------|--|---------|
| <pricingTier> | false    | The pricing tier for your Web App. The default value is <b>P1V2</b> .                  | 0.1.0+  |
| <runtime>     | true     | The runtime environment configuration, you could see the detail <a href="#">here</a> . | 0.1.0+  |
| <deployment>  | true     | The deployment configuration, you could see the details <a href="#">here</a> .         | 0.1.0+  |

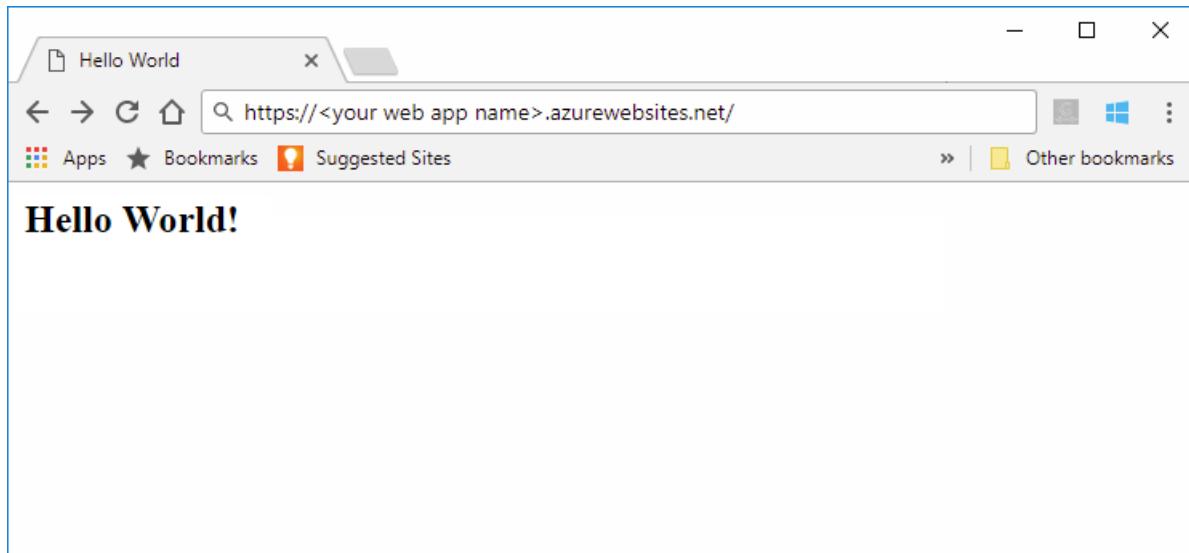
[I ran into an issue](#)

## Deploy the app

Deploy your Java app to Azure using the following command:

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

Once deployment has completed, browse to the deployed application using the following URL in your web browser, for example <http://<webapp>.azurewebsites.net>.



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

[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

[Configure Java app](#)

[CI/CD with Jenkins](#)

[Other Azure for Java Developers Resources](#)

[Learn More about Maven plugins for Azure](#)

# Quickstart: Create a Python app in Azure App Service on Linux

2/21/2020 • 5 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. The web app you configure uses a free App Service tier, so you incur no costs in the course of this article.

If you prefer to deploy apps through an IDE, see [Deploy Python apps to App Service from Visual Studio Code](#).

## Prerequisites

- Azure subscription - [create one for free](#)
- [Python 3.7](#) (Python 3.6 is also supported)
- [Git](#)
- [Azure CLI](#)

## Download the sample

In a terminal window, run the following command to clone the sample application to your local computer.

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

Then go into that folder:

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

The repository contains an *application.py* file, which tells App Service that the code contains a Flask app. For more information, see [Container startup process and customizations](#).

## Run the sample

In a terminal window, use the commands below (as appropriate for your operating system) to install the required dependencies and launch the built-in development server.

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

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
export FLASK_APP=application.py
flask run
```

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



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

## Sign in to Azure

The Azure CLI provides you with many convenient commands that you use from a local terminal to provision and manage Azure resources from the command line. You can use commands to complete the same tasks that you would through the Azure portal in a browser. You can also use CLI commands in scripts to automate management processes.

To run Azure commands in the Azure CLI, you must first sign in using the `az login` command. This command opens a browser to gather your credentials.

```
az login
```

## Deploy the sample

The `az webapp up` command creates the web app on App Service and deploys your code.

In the `python-docs-hello-world` folder that contains the sample code, run the following `az webapp up` command. Replace `<app-name>` with a globally unique app name (*valid characters are a-z, 0-9, and -*). Also replace `<location-name>` with an Azure region such as **centralus**, **eastasia**, **westeurope**, **koreasouth**, **brazilsouth**, **centralindia**, and so on. (You can retrieve a list of allowable regions for your Azure account by running the `az account list-locations` command.)

```
az webapp up --sku F1 -n <app-name> -l <location-name>
```

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

```
The behavior of this command has been altered by the following extension: webapp
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 '<app-name>' ....
Webapp creation complete
Creating zip with contents of dir /home/username/quickstart/python-docs-hello-world ...
Preparing to deploy contents to app.
All done.

{
  "app_url": "https://<app-name>.azurewebsites.net",
  "location": "Central US",
  "name": "<app-name>",
  "os": "Linux",
  "resourcegroup": "appsvc_rg_Linux_centralus",
  "serverfarm": "appsvc_asp_Linux_centralus",
  "sku": "BASIC",
  "src_path": "/home/username/quickstart/python-docs-hello-world",
  "version_detected": "-",
  "version_to_create": "python|3.7"
}
```

#### 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](#) files from the current working directory to the app.

## Browse to the app

Browse to the deployed application in your web browser at the URL `http://<app-name>.azurewebsites.net`.

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 on Linux.

## Redeploy updates

In your favorite code editor, open `application.py` and change the `return` statement on the last line to match the following code. The `print` statement is included here to generate logging output that you work with in the next section.

```
print("Handling request to home page.")  
return "Hello Azure!"
```

Save your changes and exit the editor.

Redeploy the app using the following `az webapp up` command, using the same command you used to deploy the app the first time, replacing `<app-name>` and `<location-name>` with the same names you used before.

```
az webapp up --sku F1 -n <app-name> -l <location-name>
```

Once deployment has completed, switch back to the browser window open to

`http://<app-name>.azurewebsites.net` and refresh the page, which should display the modified message:



#### 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.

First, turn on container logging by running the following command in a terminal, replacing `<app-name>` with the name of your app and `<resource-group-name>` with the name of the resource group shown in the output of the `az webapp up` command you used (such as "appsvc\_rg\_Linux\_centralus"):

```
az webapp log config --name <app-name> --resource-group <resource-group-name> --docker-container-logging  
filesystem
```

Once container logging is turned on, run the following command to show the log stream:

```
az webapp log tail --name <app-name> --resource-group <resource-group-name>
```

Refresh the app in the browser to generate console logs, which should include lines similar to the following text. If you don't see output immediately, try again in 30 seconds.

```
2019-10-23T12:40:03.815574424Z Handling request to home page.  
2019-10-23T12:40:03.815602424Z 172.16.0.1 - - [23/Oct/2019:12:40:03 +0000] "GET / HTTP/1.1" 200 12 "-"  
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/78.0.3904.63  
Safari/537.36 Edg/78.0.276.19"
```

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

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

The screenshot shows the Azure portal search interface. The search bar at the top contains the text "App Services". Below the search bar, the results are displayed under the heading "Services" with a total of "All 58 results". The results list includes various Azure services: App Services, Function App, Service Health, App Service Certificates, App Service Domains, App Service Environments, App Service plans, Lab Services, Media services, and Bot Services. Each service item has a small icon to its left.

Select the name of your Azure app.

The screenshot shows the "App Services" blade in the Azure portal. At the top, there are buttons for "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", "Stop", and "Delete". Below these, 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":

| Name                     | Status  | App Type | App Service Plan           | Location   | Subscription         |
|--------------------------|---------|----------|----------------------------|------------|----------------------|
| <a href="#">App Name</a> | Running | Web App  | user_asp_Linux_centralus_0 | Central US | SomeSubscriptionName |

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

The screenshot shows the "Overview" page for the "App Name" app service. The left sidebar has links for "Overview", "Activity log", "Access control (IAM)", "Tags", "Diagnose and solve problems", "Security", "Deployment", "Quickstart", "Deployment slots", and "Deployment Center". The main content area displays the following details:

| Resource group (change)    | user_rg_Linux_centralus   |
|----------------------------|---|
| Status                     | Running   |
| Location                   | Central US  |
| Subscription (change)      | SomeSubscriptionName  |
| Subscription ID            | 00000000-0000-0000-0000-000000000000  |
| Tags (change)              | cli : webapp_up   |
| URL                        | <a href="https://&lt;app name&gt;.azurewebsites.net">https://&lt;app name&gt;.azurewebsites.net</a>                         |
| App Service Plan           | user_asp_Linux_centralus_0 (P1v2: 1)  |
| FTP/deployment username    |   |
| No FTP/deployment user set |   |
| FTP hostname               | <a href="ftp://waws-prod-dm1-123.ftp.azurewebsites.windows.net">ftp://waws-prod-dm1-123.ftp.azurewebsites.windows.net</a>   |
| FTPS hostname              | <a href="ftps://waws-prod-dm1-123.ftp.azurewebsites.windows.net">ftps://waws-prod-dm1-123.ftp.azurewebsites.windows.net</a> |

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

## 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 use an App Service SKU other than the free F1

tier, these resources will incur ongoing costs.

If you don't expect to need these resources in the future, delete the resource group by running the following command, replacing <resource-group-name> with the resource group shown in the output of the az webapp up command, such as "appsvc\_rg\_Linux\_centralus". The command may take a minute to complete.

```
az group delete -n <resource-group-name>
```

## Next steps

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

[Configure Python app](#)

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

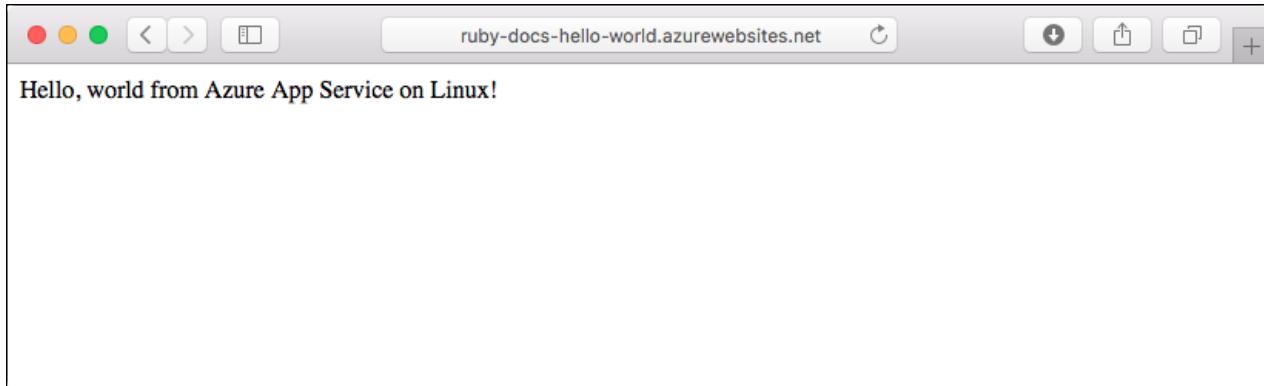
# Create a Ruby on Rails App in App Service on Linux

2/21/2020 • 6 minutes to read • [Edit Online](#)

[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 Azure 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

In a terminal window, run the following command to clone the sample app repository to your local machine:

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

## Run the application locally

Run the application locally so that you see how it should look when you deploy it to Azure. Open a terminal window, change to the `hello-world` directory, and use the `rails server` command to start the server.

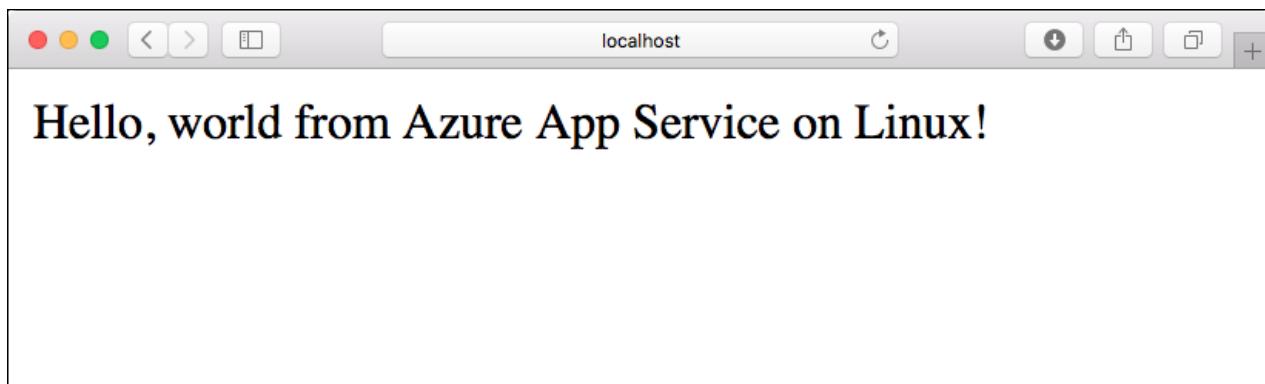
The first step is to install the required gems. There's a `Gemfile` included in the sample, so just run the following command:

```
bundle install
```

Once the gems are installed, we'll use bundler to start the app:

```
bundle exec rails server
```

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 <b>Try It</b> in the upper-right corner of a code block.<br>Selecting <b>Try It</b> doesn't automatically copy the code to Cloud Shell.            | A screenshot of a code block. At the top right, there is a green button labeled 'Try It' with a white play icon. To its left is a 'Copy' button with a clipboard icon. The entire 'Try It' button is highlighted with a red box. |
| Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. | A screenshot of the Azure portal's blue header menu. On the far left is a 'Cloud Shell' button, which is highlighted with a red box. To its right are other icons for 'Compute', 'Logs', 'Metrics', and a question mark.         |
| Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .   | A screenshot of the Azure portal's blue header menu. On the far left is a 'Cloud Shell' button, which is highlighted with a red box. To its right are other icons for 'Compute', 'Logs', 'Metrics', and a question mark.         |

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 like 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 in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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|2.3`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"RUBY|2.6.2" --deployment-local-git
# PowerShell
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

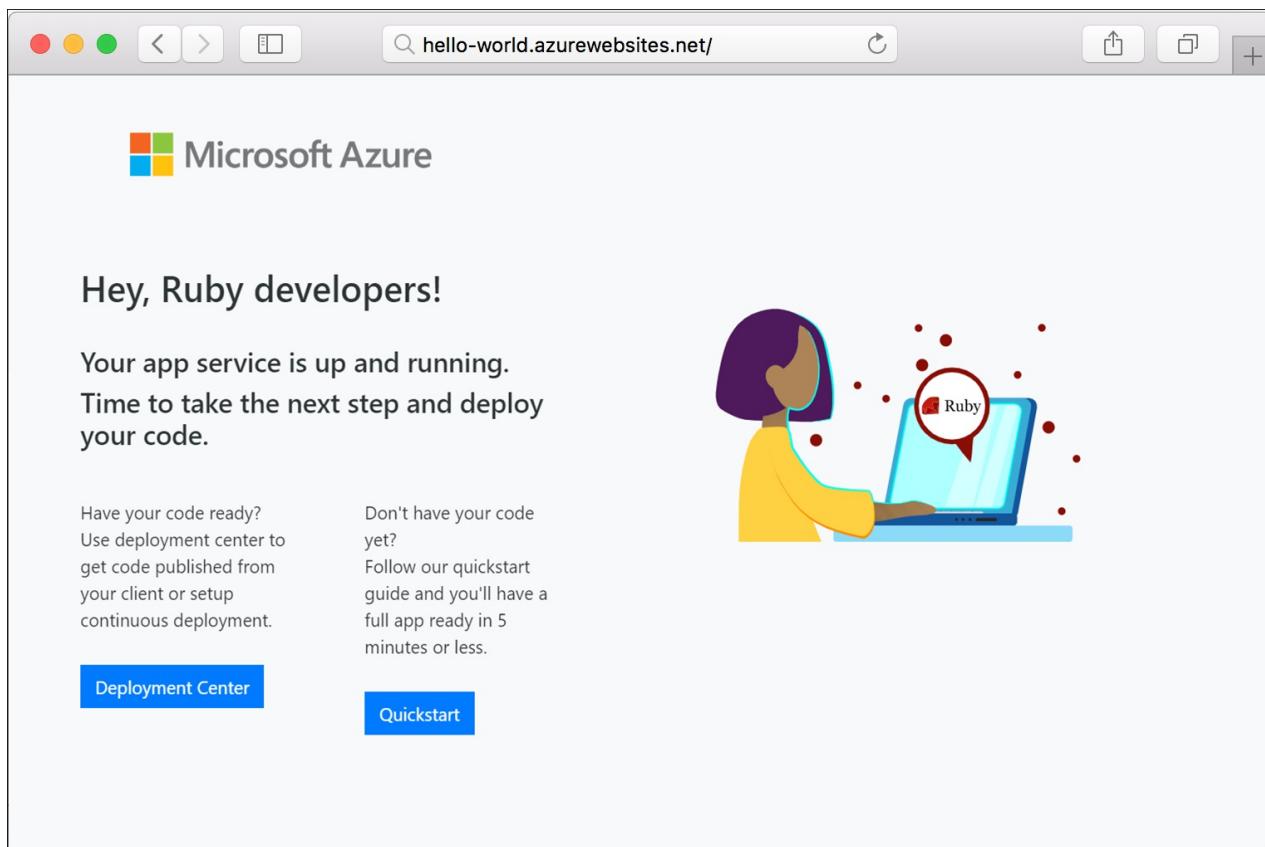
The URL of the Git remote is shown in the `deploymentLocalGitUrl` property, with the format

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

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

Run the following commands to deploy the local application to your Azure web app:

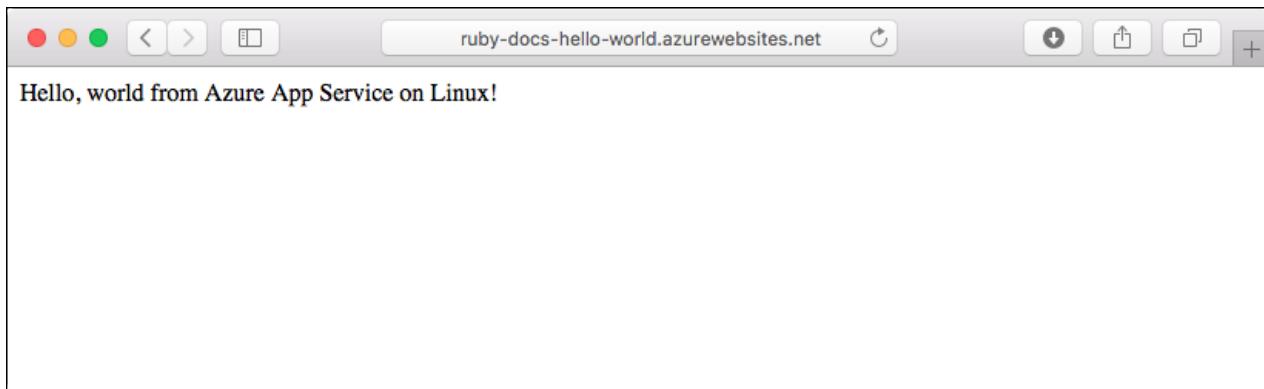
```
git remote add azure <Git deployment URL from above>
git push azure master
```

Confirm that the remote deployment operations report success. The commands produce output similar to the following text:

```
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  master -> master
```

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](#)

# Deploy a custom Linux container to Azure App Service

12/12/2019 • 3 minutes to read • [Edit Online](#)

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 an image

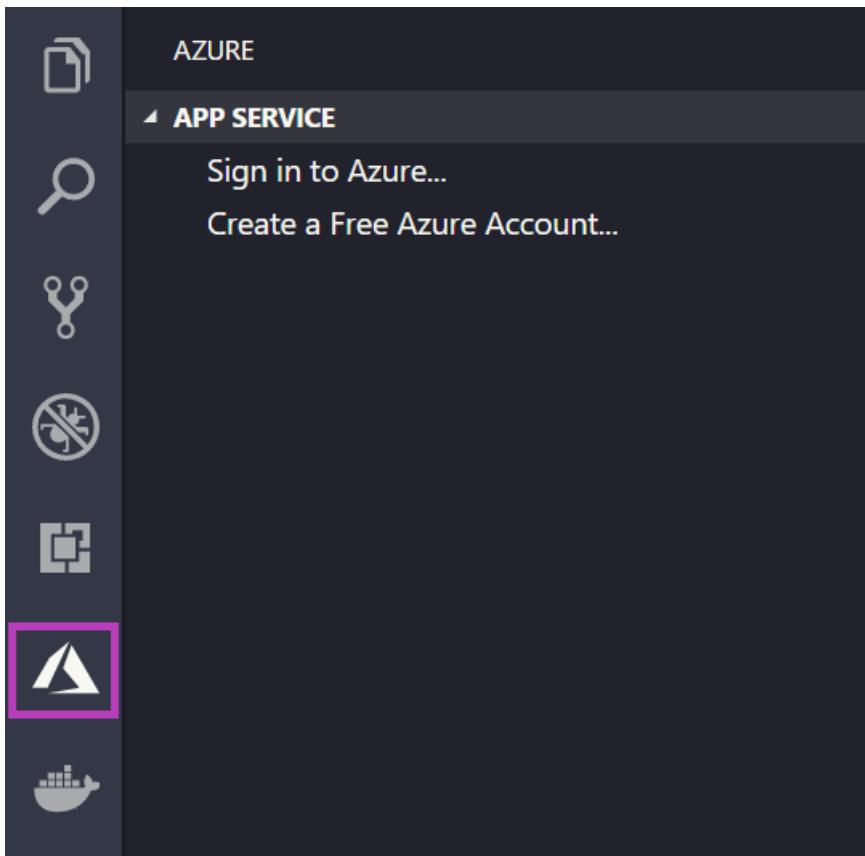
To complete this quickstart, you will need a suitable web app image stored in an [Azure Container Registry](#). Follow the instructions in [Quickstart: Create a private container registry using the Azure portal](#), but use the `mcr.microsoft.com/azuredocs/go` image instead of the `hello-world` image. For reference, the [sample Dockerfile](#) is found in [Azure Samples repo](#).

### IMPORTANT

Be sure to set the **Admin User** option to **Enable** when you create the 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.

## Sign in

Next, launch VS Code and log into your Azure account using the App Service extension. To do this, select the Azure logo in the Activity Bar, navigate to the **APP SERVICE** explorer, then select **Sign in to Azure** and follow the instructions.



## Check prerequisites

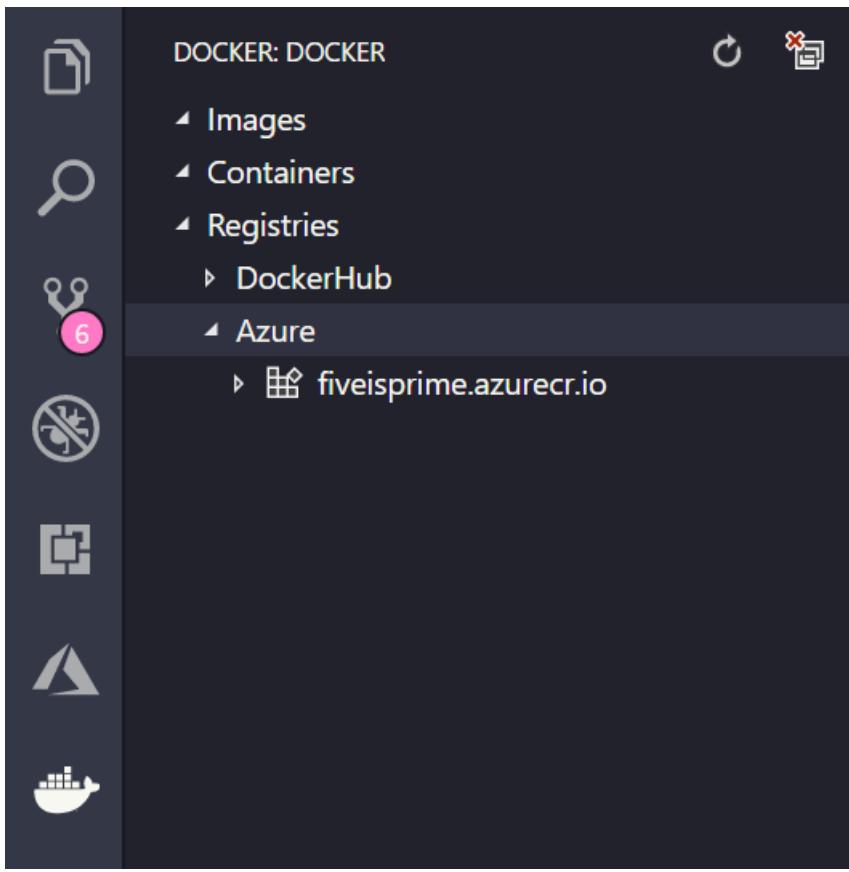
Now you can check whether you have all the prerequisites installed and configured properly.

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

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

```
docker --version
```

Finally, ensure that your Azure Container Registry is connected. To do this, select the Docker logo in the Activity Bar, then navigate to **REGISTRIES**.



## Deploy the image to Azure App Service

Now that everything is configured, you can deploy your image to [Azure App Service](#) directly from the Docker extension explorer.

Find the image under the **Registries** node in the **DOCKER** explorer, and expand it to show its tags. Right-click a tag and then select **Deploy Image to Azure App Service**.

From here, 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.

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.

## Browse the website

The **Output** panel will open during deployment to indicate the status of the operation. When the operation completes, find the app you created in the **APP SERVICE** explorer, right-click it, then select **Browse Website** to open the site in your browser.

[I ran into an issue](#)

## Next steps

Congratulations, you've successfully completed this quickstart!

Next, check out the other Azure extensions.

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

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

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

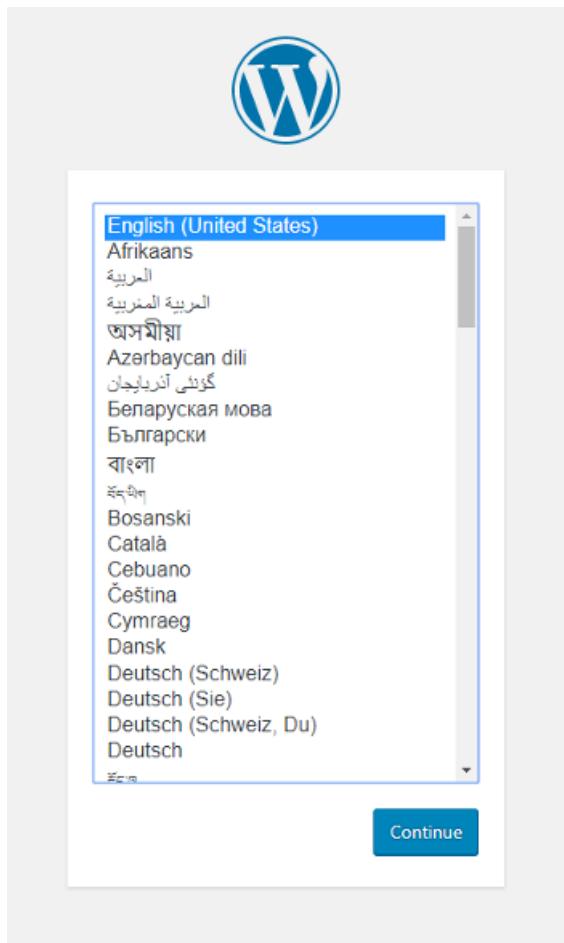
12/12/2019 • 3 minutes to read • [Edit Online](#)

## 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.

You'll complete this quickstart in Cloud Shell, but you can also run these commands locally with [Azure CLI](#) (2.0.32 or later).



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 <b>Try It</b> in the upper-right corner of a code block.<br>Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |  |
| Select the <b>Cloud Shell</b> 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.

## 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 like 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

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 `-`).

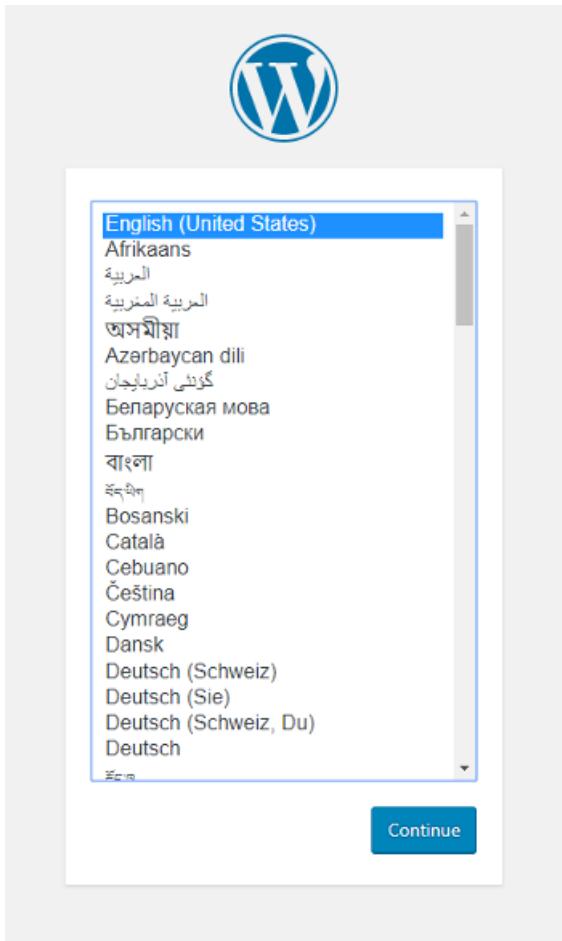
```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app_name> --multicontainer-  
config-type compose --multicontainer-config-file compose-wordpress.yml
```

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](#)

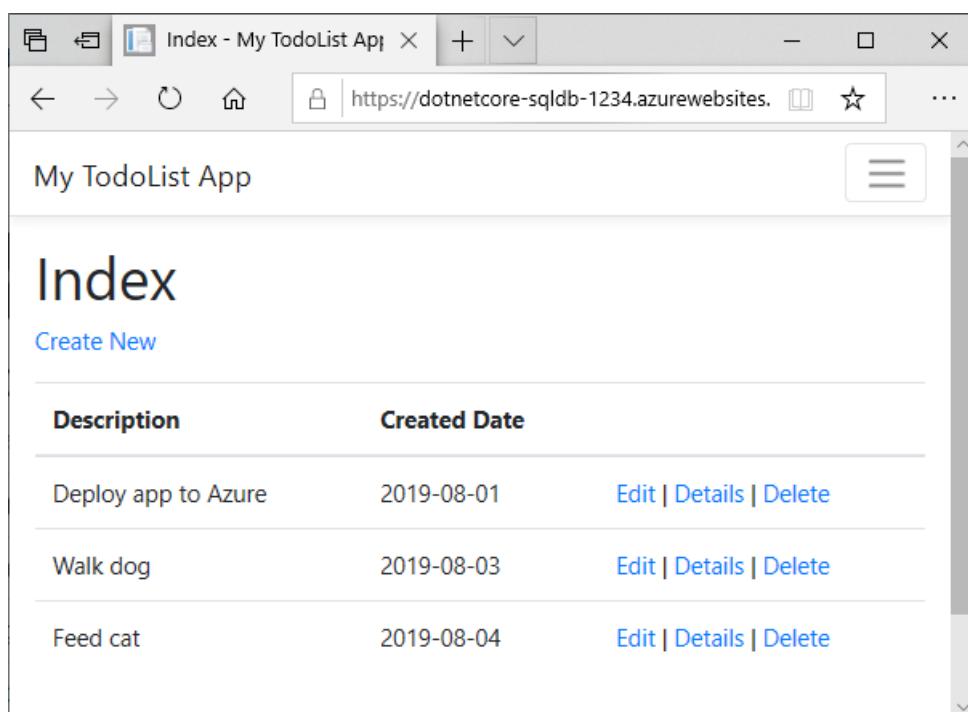
# Build an ASP.NET Core and SQL Database app in Azure App Service on Linux

2/21/2020 • 13 minutes to read • [Edit Online](#)

## NOTE

This article deploys an app to App Service on Linux. To deploy to App Service on *Windows*, see [Build a .NET Core and SQL Database app in Azure App Service](#).

[App Service on Linux](#) provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create a .NET Core app and connect it to a SQL Database. When you're done, you'll have a .NET Core MVC app running in App Service on Linux.



In this tutorial, you learn how to:

- Create a SQL Database in Azure
- Connect a .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 .NET Core SDK 2.2](#)

# Create local .NET Core app

In this step, you set up the local .NET Core project.

## Clone the sample application

In the terminal window, `cd` to a working directory.

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](#).

## Run the application

Run the following commands to install the required packages, run database migrations, and start the application.

```
dotnet restore  
dotnet ef database update  
dotnet run
```

Navigate to `http://localhost:5000` in a browser. Select the **Create New** link and create a couple *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 |

To stop .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 <b>Try It</b> in the upper-right corner of a code block.<br>Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser. |              |
| Select the <b>Cloud Shell</b> 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.

## 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 like 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 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 SQL Database endpoint, `<server-name>.database.windows.net`, so the name needs to be unique across all logical servers in Azure. The name must contain only lowercase letters, numbers, and the hyphen (-) character, and must be between 3 and 50 characters long. 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": "sqladmin",  
  "administratorLoginPassword": null,  
  "fullyQualifiedDomainName": "<server-name>.database.windows.net",  
  "id": "/subscriptions/00000000-0000-0000-0000-  
00000000/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

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
```

## 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
```

## Create connection string

Replace the following string with the `<server-name>`, `<db-username>`, and `<db-password>` you used earlier.

```
Server=tcp:<server-name>.database.windows.net,1433;Database=coreDB;User ID=<db-username>;Password=<db-  
password>;Encrypt=true;Connection Timeout=30;
```

This is the connection string for your .NET Core app. Copy it for use later.

## Deploy app to Azure

In this step, you deploy your SQL Database-connected .NET Core application to App Service on Linux.

### 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 in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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 `DOTNETCORE|2.2`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"DOTNETCORE|2.2" --deployment-local-git
# PowerShell
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"DOTNETCORE|2.2" --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 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
```

### 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 SQLServer
```

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 [Connect to SQL Database in production](#).

### Configure environment variable

Next, set `ASPNETCORE_ENVIRONMENT` app setting to *Production*. This setting lets you know whether you're running in Azure, because you use SQLite for your local development environment and SQL Database for your Azure environment.

The following example configures a `ASPNETCORE_ENVIRONMENT` app setting in your Azure app. Replace the `<app-name>` placeholder.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings  
ASPNETCORE_ENVIRONMENT="Production"
```

To see how the environment variable is referenced in your code, see [Connect to SQL Database in production](#).

### Connect to SQL Database in production

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, which uses the environment variables that you configured earlier.

```
// Use SQL Database if in Azure, otherwise, use SQLite
if(Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT") == "Production")
    services.AddDbContext<MyDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("MyDbConnection")));
else
    services.AddDbContext<MyDbContext>(options =>
        options.UseSqlite("Data Source=MvcMovie.db"));

// Automatically perform database migration
services.BuildServiceProvider().GetService<MyDbContext>().Database.Migrate();
```

If this code detects that it's running in production (which indicates the Azure environment), then it uses the connection string you configured to connect to the SQL Database. For information on how app settings are accessed in App Service, see [Access environment variables](#).

The `Database.Migrate()` call helps you when it's run in Azure, because it automatically creates the databases that your .NET Core app needs, based on its migration configuration.

Save your changes, then commit it into your Git repository.

```
git add .
git commit -m "connect to SQLDB in Azure"
```

## Push to Azure from Git

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>
```

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 master
```

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

```
Counting objects: 98, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (92/92), done.  
Writing objects: 100% (98/98), 524.98 KiB | 5.58 MiB/s, done.  
Total 98 (delta 8), reused 0 (delta 0)  
remote: Updating branch 'master'.  
remote: .  
remote: Updating submodules.  
remote: Preparing deployment for commit id '0c497633b8'.  
remote: Generating deployment script.  
remote: Project file path: ./DotNetCoreSqlDb.csproj  
remote: Generated deployment script files  
remote: Running deployment command...  
remote: Handling ASP.NET Core Web Application deployment.  
remote: .  
remote: .  
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  
 * [new branch]      master -> master
```

## Browse to the Azure app

Browse to the deployed app using your web browser.

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

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 .NET Core app in App Service on Linux.

## 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; }
```

## Run Code First Migrations locally

Run a few commands to make updates to your local database.

```
dotnet ef migrations add AddProperty
```

Update the local database:

```
dotnet ef database update
```

## 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.

Open `Controllers\TodosController.cs`.

Find the `Create()` 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)
```

Open `Views\Todos\Create.cshtml`.

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" />
        <span asp-validation-for="Done" class="text-danger"></span>
    </div>
</div>
```

Open `Views\Todos\Index.cshtml`.

Search for the empty `<th></th>` element. Just above this element, add the following Razor code:

```
<th>
    @Html.DisplayNameFor(model => model.Done)
</th>
```

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

Run the app locally.

```
dotnet run
```

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

```
git add .
git commit -m "added done field"
git push azure master
```

Once the `git push` is complete, navigate to your Azure app and try out the new functionality.

Description	Created Date	Done	
Deploy app to Azure	2019-08-01	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Walk dog	2019-08-03	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Feed cat	2019-08-04	<input type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Check email	2019-08-05	<input checked="" type="checkbox"/>	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>

All your existing to-do items are still displayed. When you republish your .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

The sample project already follows the guidance at [ASP.NET Core Logging in Azure](#) with two configuration changes:

- Includes a reference to `Microsoft.Extensions.Logging.AzureAppServices` in `DotNetCoreSqlDb.csproj`.
- Calls `loggerFactory.AddAzureWebAppDiagnostics()` in `Startup.cs`.

#### NOTE

The project's log level is set to `Information` in `appsettings.json`.

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

#### 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 customizing the ASP.NET Core logs, see [Logging in ASP.NET Core](#).

## 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 blade. On the left, there is a vertical navigation bar with icons for Home, App Services, Functions, Logic Apps, Container Registry, and App Configuration. The 'App Services' icon is highlighted with a red box. The main area has a dark blue header with the title 'App Services' and the resource group name 'rickaOrgName'. Below the header, there are buttons for 'Add', 'Columns', and 'Refresh'. A search bar labeled 'Filter by name...' and dropdown menus for 'All subscriptions', 'All locations', and 'No grouping' are present. A summary section shows '1 items' found. A table lists the app details: NAME (DotNetAppSqlDb1234), RESOURCE GR... (myResourceGroup), STATUS (Running), APP TYPE (Web app), and APP SERVICE PLAN (myAppServicePlan). The row for 'DotNetAppSqlDb1234' is also highlighted with a red box.

NAME	RESOURCE GR...	STATUS	APP TYPE	APP SERVICE PLAN
DotNetAppSqlDb1234	myResourceGroup	Running	Web app	myAppServicePlan

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 portal interface for managing an App Service named 'DotNetAppSqlDb1234'. The left sidebar contains a navigation menu with various icons and links. The main content area is divided into sections: 'Essentials' (Resource group, Status, Location, Subscription), 'Monitoring' (Requests and errors chart), and other monitoring and diagnostic tools.

## 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 .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](#)

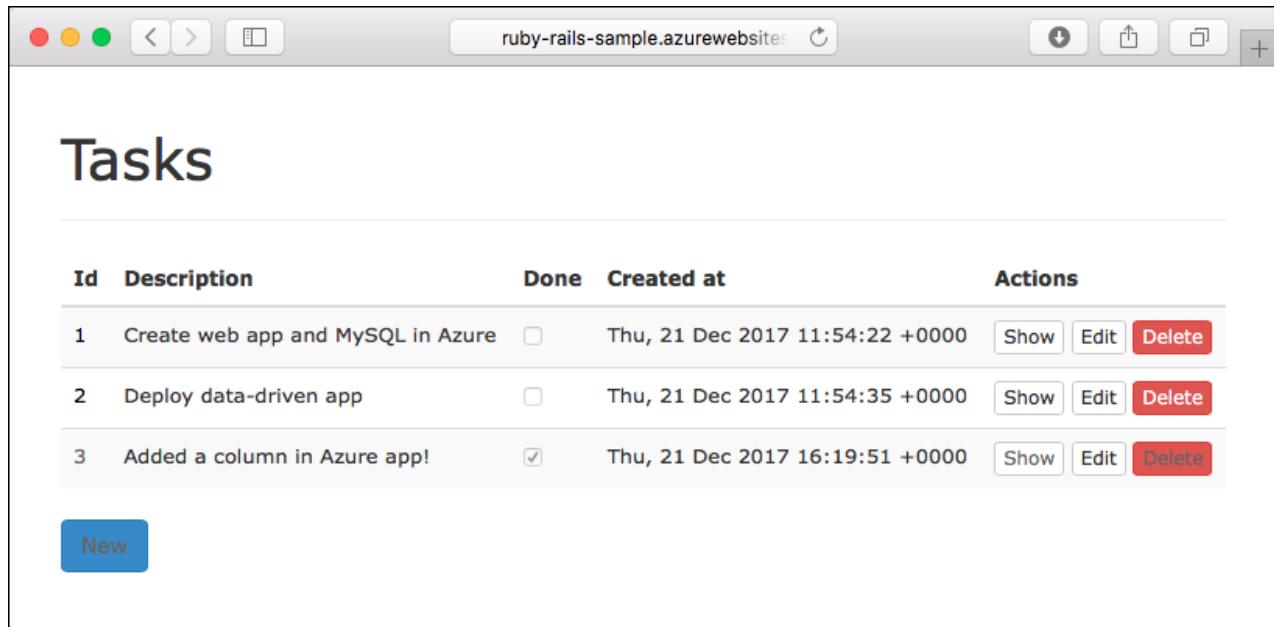
Or, check out other resources:

[Configure ASP.NET Core app](#)

# Build a Ruby and Postgres app in Azure App Service on Linux

2/21/2020 • 13 minutes to read • [Edit Online](#)

[App Service on Linux](#) provides a highly scalable, self-patching web hosting service using the Linux operating system. 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 web browser window with the URL `ruby-rails-sample.azurewebsites.net` in the address bar. The page title is "Tasks". Below the title is a table listing three 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

A blue "New" button is located at the bottom left of the table.

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 2.3](#)
- [Install Ruby on Rails 5.1](#)
- [Install and run PostgreSQL](#)

## 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

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](#).

Type `\q` to exit the Postgres client.

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

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/rubyrails-tasks.git
```

`cd` to your cloned directory. Install the required packages.

```
cd rubyrails-tasks  
bundle install --path vendor/bundle
```

### Run the sample locally

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
```

Run the application.

```
rails server
```

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

To stop the Rails server, type `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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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.

## 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 like 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 a Postgres server

Create a PostgreSQL server with the `az postgres server create` command.

Run the following command in the Cloud Shell, and substitute a unique server name for the *<postgres-server-name>* placeholder. The server name needs to be unique across all servers in Azure.

```
az postgres server create --location "West Europe" --resource-group myResourceGroup --name <postgres-server-name> --admin-user adminuser --admin-password My5up3r$tr0ngPa$w0rd! --sku-name GP_Gen4_2
```

When the Azure Database for PostgreSQL server is created, the Azure CLI shows information similar to the following example:

```
{
  "administratorLogin": "adminuser",
  "earliestRestoreDate": "2018-06-15T12:38:25.280000+00:00",
  "fullyQualifiedDomainName": "<postgres-server-name>.postgres.database.azure.com",
  "id": "/subscriptions/00000000-0000-0000-0000-
  0000000000/resourceGroups/myResourceGroup/providers/Microsoft.DBforPostgreSQL/servers/<postgres-server-
  name>",
  "location": "westeurope",
  "name": "<postgres-server-name>",
  "resourceGroup": "myResourceGroup",
  "sku": {
    "capacity": 2,
    "family": "Gen4",
    "name": "GP_Gen4_2",
    "size": null,
    "tier": "GeneralPurpose"
  },
  < Output has been truncated for readability >
}
```

### Configure server firewall

In the Cloud Shell, create a firewall rule for your Postgres server to allow client connections by using the `az postgres 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. Substitute a unique server name for the *<postgres-server-name>* placeholder.

```
az postgres server firewall-rule create --resource-group myResourceGroup --server <postgres-server-name> --
  name AllowAllIps --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](#).

## Connect to production Postgres server locally

In the Cloud Shell, connect to the Postgres server in Azure. Use the value you specified previously for the <postgres-server-name> placeholders.

```
psql -U adminuser@<postgres-server-name> -h <postgres-server-name>.postgres.database.azure.com postgres
```

When prompted for a password, use *My5up3r\$tr0ngPa\$w0rd!*, which you specified when you created the database server.

## Create a production database

At the `postgres` prompt, create a database.

```
CREATE DATABASE sampledb;
```

## Create a user with permissions

Create a database user called *railsappuser* and give it all privileges in the `sampledb` database.

```
CREATE USER railsappuser WITH PASSWORD 'MyPostgresAzure2017';
GRANT ALL PRIVILEGES ON DATABASE sampledb TO railsappuser;
```

Exit the server connection by typing `\q`.

## 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

Back in the local terminal, set the following environment variables:

```
export DB_HOST=<postgres-server-name>.postgres.database.azure.com
export DB_DATABASE=sampledb
export DB_USERNAME=railsappuser@<postgres-server-name>
export DB_PASSWORD=MyPostgresAzure2017
```

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
```

When running in the production environment, the Rails application needs precompiled assets. Generate the required assets with the following command:

```
rake assets:precompile
```

The Rails production environment also uses secrets to manage security. Generate a secret key.

```
rails secret
```

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>
```

Enable the Rails production environment to serve JavaScript and CSS files.

```
export RAILS_SERVE_STATIC_FILES=true
```

Run the sample application in the production environment.

```
rails server -e production
```

Navigate to <http://localhost:3000>. If the page loads without errors, the Ruby on Rails application is connecting to the Postgres database in Azure.

Add a few tasks in the page.

<b>Id</b>	<b>Description</b>	<b>Created at</b>	<b>Actions</b>
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

To stop the Rails server, type `ctrl + c` in the terminal.

**Commit your changes**

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 in the resource group with the [az appservice plan create](#) command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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|2.3`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash  
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime  
"RUBY|2.6.2" --deployment-local-git  
# PowerShell  
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`. 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 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="railsappuser@<postgres-server-name>" DB_PASSWORD="MyPostgresAzure2017"
```

## Configure Rails environment variables

In the local terminal, [generate a new secret](#) for the Rails production environment in Azure.

```
rails secret
```

Configure the variables required by Rails production environment.

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

In the local terminal, add an Azure remote to your local Git repository.

```
git remote add azure <paste-copied-url-here>
```

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 master
```

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 'master'.  
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	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Delete</a>
2	Deploy data-driven app	Thu, 21 Dec 2017 11:54:35 +0000	<a href="#">Show</a> <a href="#">Edit</a> <a href="#">Delete</a>

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

In the terminal, navigate to the root of the Git repository.

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.

In the terminal, run Rails database migrations to make the change in the local database.

```
rake db:migrate
```

### Update application logic

Open the `app/controllers/tasks_controller.rb` file. At the end of the file, find the following line:

```
params.require(:task).permit(:Description)
```

Modify this line to include the new `Done` parameter.

```
params.require(:task).permit(:Description, :Done)
```

### Update the views

Open the `app/views/tasks/_form.html.erb` file, which is the Edit form.

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>
```

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><strong><%= model_class.human_attribute_name(:Done) %></strong></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>
```

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

In the local terminal, run the Rails server.

```
rails server
```

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

To stop the Rails server, type `Ctrl + C` in the terminal.

## Publish changes to Azure

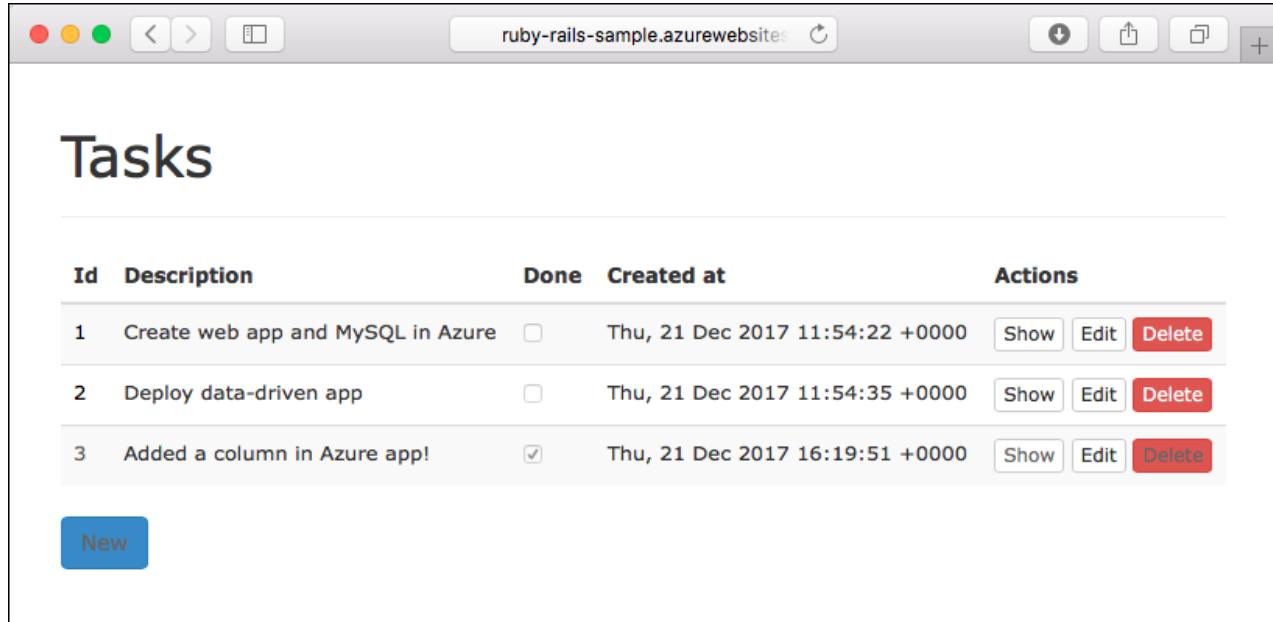
In the terminal, run Rails database migrations for the production environment to make the change in the Azure database.

```
rake db:migrate RAILS_ENV=production
```

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 master
```

Once the `git push` is complete, navigate to the Azure app and test the new functionality.



The screenshot shows a web browser window with the title bar "ruby-rails-sample.azurewebsites.net". The main content area displays a "Tasks" page. At the top, there's a heading "Tasks". Below it is a table with three rows of task data:

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, there's a blue "New" button.

If you added any tasks, they are retained in the database. Updates to the data schema leave existing data intact.

## Stream diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

### NOTE

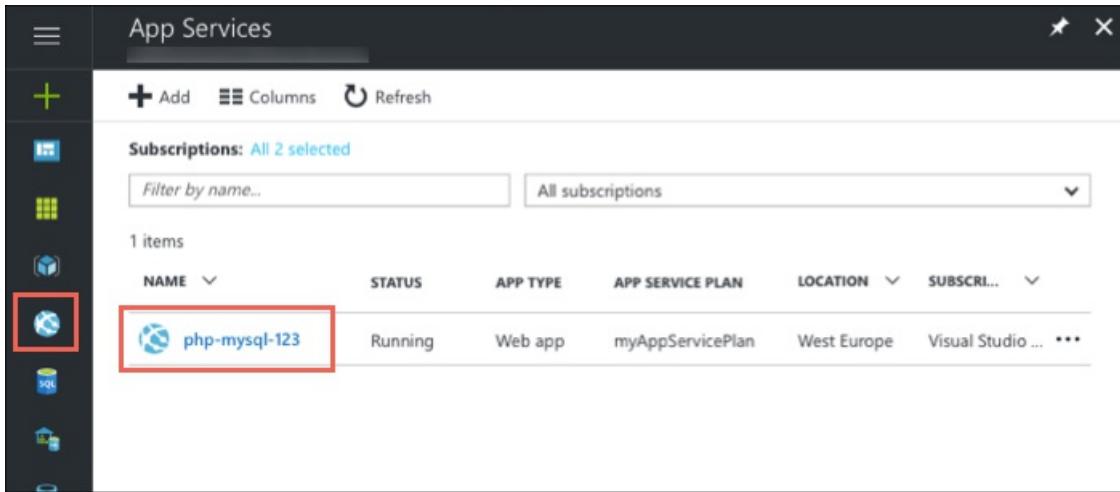
You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

# Manage the Azure app

Go to the [Azure portal](#) to manage the app you created.

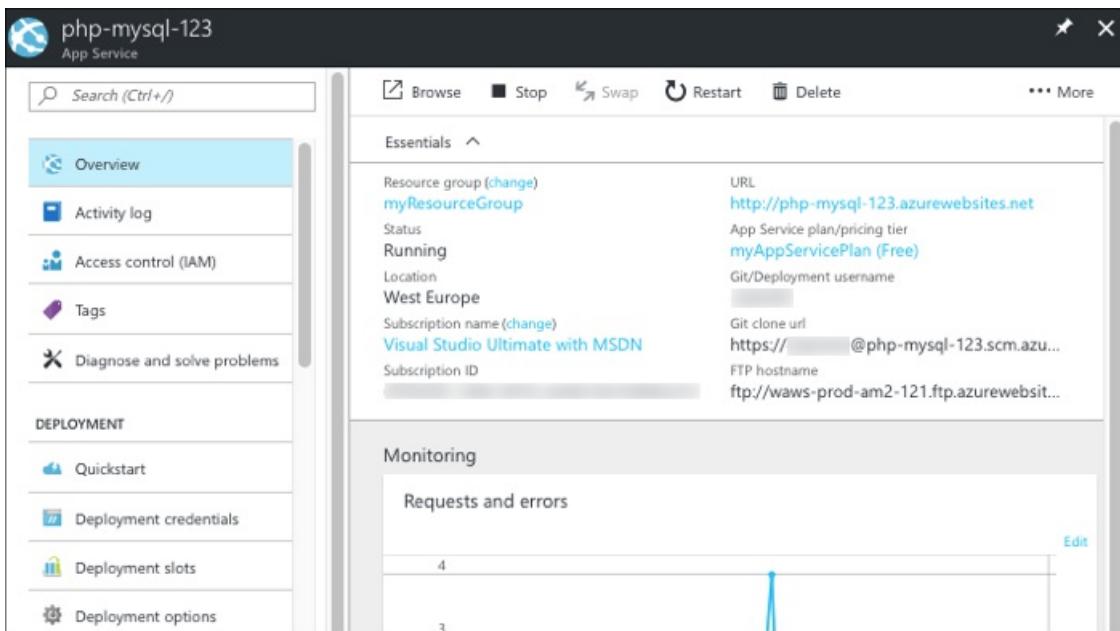
From the left menu, click **App Services**, and then click the name of your Azure app.



NAME	STATUS	APP TYPE	APP SERVICE PLAN	LOCATION	SUBSCR...
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.



## 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](#)

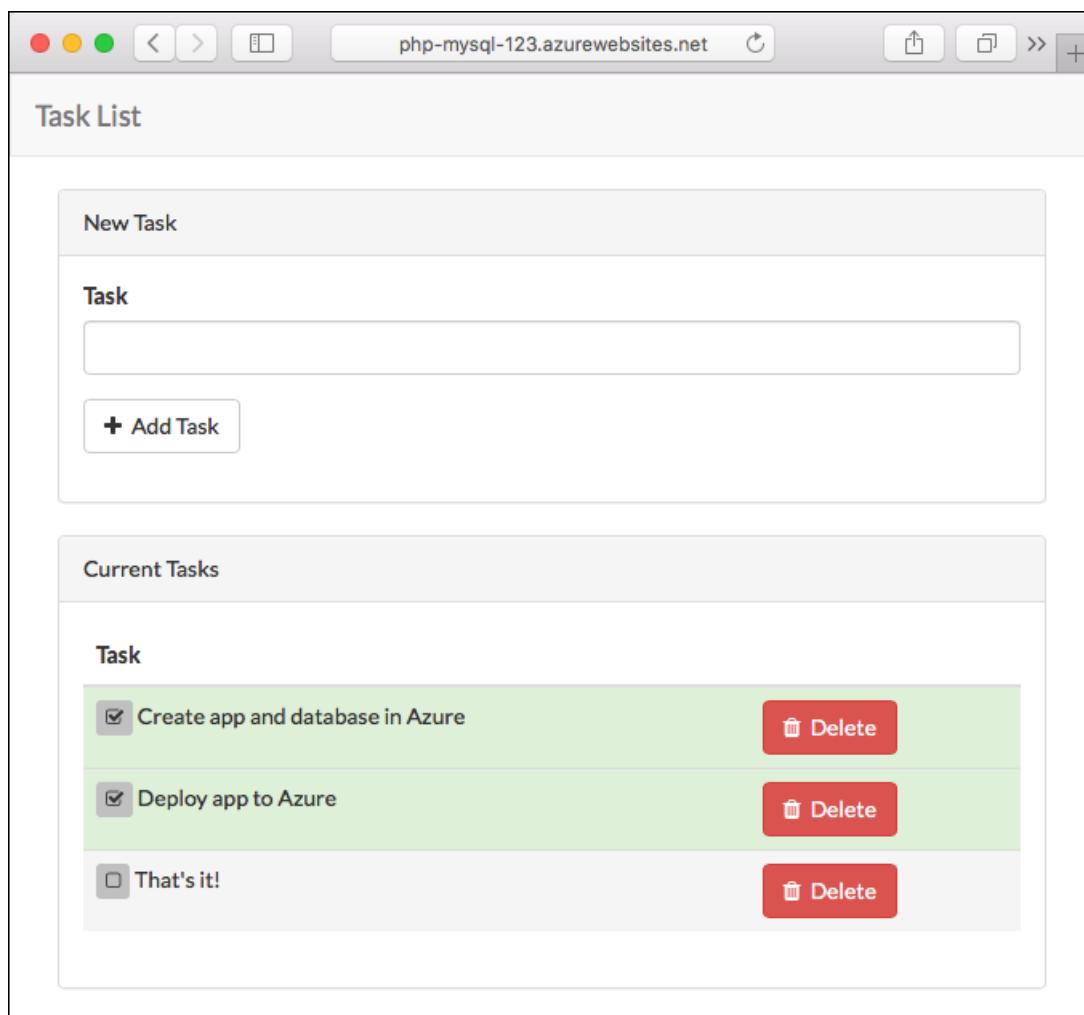
# Build a PHP and MySQL app in Azure App Service on Linux

2/28/2020 • 16 minutes to read • [Edit Online](#)

## NOTE

This article deploys an app to App Service on Linux. To deploy to App Service on *Windows*, see [Build a PHP and MySQL app in Azure](#).

[App Service on Linux](#) provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create a PHP app and connect it to a MySQL database. When you're finished, you'll have a [Laravel](#) app running on 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](#)

## 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

At the `mysql` prompt, create a database.

```
CREATE DATABASE sampledb;
```

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.

Run the following command to clone the sample repository.

```
git clone https://github.com/Azure-Samples/laravel-tasks
```

`cd` to your cloned directory. Install the required packages.

```
cd laravel-tasks  
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=SomeRandomString  
  
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

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
```

Generate a new Laravel application key.

```
php artisan key:generate
```

Run the application.

```
php artisan serve
```

Navigate to `http://localhost:8000` in a browser. Add a few tasks in the page.

The screenshot shows a web application window titled "Task List". At the top, there's a header bar with standard browser controls (back, forward, search, etc.) and a title "localhost". Below the header, the main content area has two sections:

- New Task**: A form with a text input field labeled "Task" and a button labeled "+ Add Task".
- Current Tasks**: A list of three items:
  - Create app and database in Azure Delete
  - Deploy data-driven app Delete
  - That's it! Delete

To stop PHP, type `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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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.

## 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 like 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 a MySQL server

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, please 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",
  ...
}
```

### 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](#).

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>
```

### Connect to production MySQL server locally

In the 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
```

### Create a production database

At the `mysql` prompt, create a database.

```
CREATE DATABASE sampledb;
```

### Create a user with permissions

Create a database user called `phpappuser` and give it all privileges in the `sampledb` database.

```
CREATE USER 'phpappuser' IDENTIFIED BY 'MySQLAzure2017';
GRANT ALL PRIVILEGES ON sampledb.* TO 'phpappuser';
```

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>*.

```
APP_ENV=production
APP_DEBUG=true
APP_KEY=SomeRandomString

DB_CONNECTION=mysql
DB_HOST=<mysql-server-name>.mysql.database.azure.com
DB_DATABASE=sampledbs
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 SSL certificate

By default, Azure Database for MySQL enforces SSL 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') && 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

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
```

`.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
```

Run the sample application with `.env.production` as the environment file.

```
php artisan serve --env=production
```

Navigate to `http://localhost:8000`. If the page loads without errors, the PHP application is connecting to the MySQL database in Azure.

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 labeled "Task" and a button labeled "+ Add Task".
- Current Tasks:** Contains three entries:
  - "Create app and database in Azure" with a "Delete" button.
  - "Deploy data-driven app" with a "Delete" button.
  - "That's it!" with a "Delete" button.

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.

The Laravel application starts in the `/public` 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](#).

### 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 in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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|7.0`. 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|7.0" --deployment-local-git
# PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PHP|7.0" --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 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.

The following command configures the app settings `DB_HOST`, `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD`. Replace the placeholders `<appname>` 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 app 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.

Use `php artisan` to generate a new application key without saving it to `.env`.

```
php artisan key:generate --show
```

Set the application key in the App Service app by using the `az webapp config appsettings set` command. Replace the placeholders `<appname>` 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.

## Push to Azure from Git

Add an Azure remote to your local Git repository.

```
git remote add azure <paste_copied_url_here>
```

Push to the Azure remote to deploy the PHP application. You are prompted for the password you supplied earlier as part of the creation of the deployment user.

```
git push azure master
```

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 'master'.  
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.

The screenshot shows a web application interface for managing tasks. At the top, there are standard browser controls (back, forward, search, etc.) and a title bar displaying the URL 'php-mysql-123.azurewebsites.net'. Below the title bar, the page is titled 'Task List'. There are two main sections: 'New Task' and 'Current Tasks'. The 'New Task' section contains a text input field labeled 'Task' and a button labeled '+ Add Task'. The 'Current Tasks' section lists three items: 'Create app and database in Azure', 'Deploy app to Azure', and 'That's it!'. Each item has a red 'Delete' button to its right.

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

In the terminal, navigate to the root of the Git repository.

Generate a new database migration for the `tasks` table:

```
php artisan make:migration add_complete_column --table=tasks
```

This command shows you the name of the migration file that's generated. Find this file in `database/migrations` and open it.

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 terminal, 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

From the root directory of the Git repository, run the development server.

```
php artisan serve
```

To see the task status change, navigate to <http://localhost:8000> and select the checkbox.

The screenshot shows a web application interface for managing tasks. At the top, there's a browser-like header with icons for window control and a search bar labeled "localhost". Below this, the main content area is titled "Task List". On the left, there's a "New Task" section with a text input field and a button labeled "+ Add Task". On the right, under "Current Tasks", there's a list of three tasks:

- "Create app and database in Azure" - The checkbox is checked, and the background is green. To its right is a red "Delete" button with a trash icon.
- "Deploy data-driven app" - The checkbox is checked, and the background is green. To its right is a red "Delete" button with a trash icon.
- "That's it!" - The checkbox is unchecked, and the background is grey. To its right is a red "Delete" button with a trash icon.

To stop PHP, type `Ctrl + C` in the terminal.

### Publish changes to Azure

In the terminal, run Laravel database migrations with the production connection string to make the change in the Azure database.

```
php artisan migrate --env=production --force
```

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 master
```

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 a local host. The "New Task" section is empty. The "Current Tasks" section contains three items:

- Create app and database in Azure (checked) - Delete button
- Deploy app to Azure (checked) - Delete button
- That's it! (unchecked) - Delete button

If you added any tasks, they are retained in the database. Updates to the data schema leave existing data intact.

## Stream diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

## NOTE

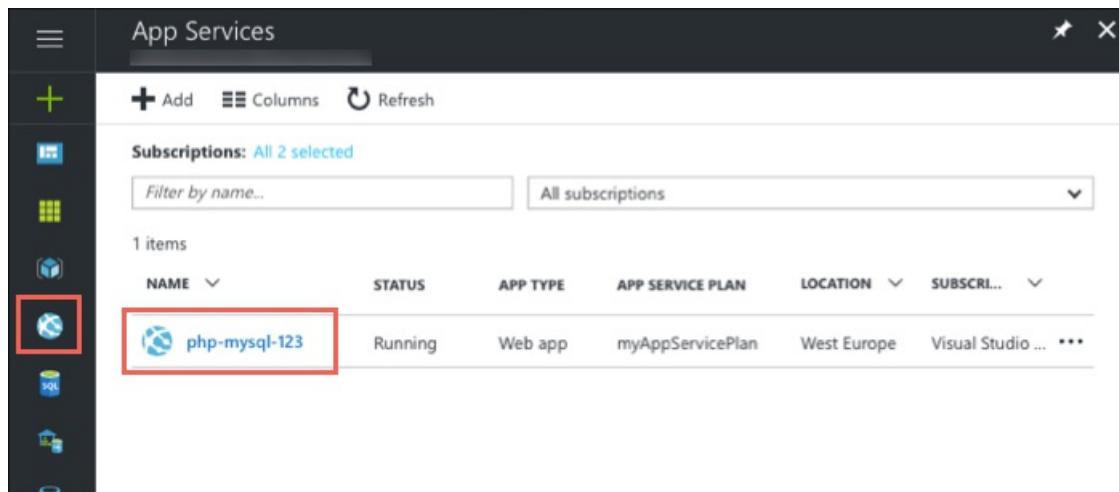
You can also inspect the log files from the browser at <https://<app-name>.scm.azurewebsites.net/api/logs/docker>.

To stop log streaming at any time, type `Ctrl + C`.

## Manage the Azure app

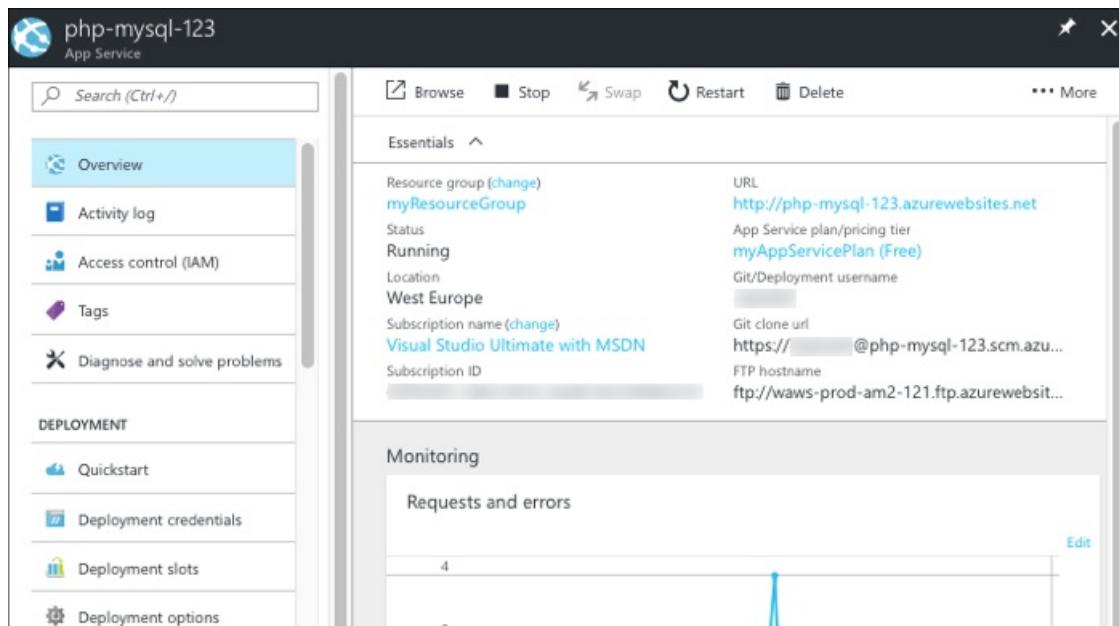
Go to the [Azure portal](#) to manage the app you created.

From the left menu, click **App Services**, and then click the name of your Azure app.



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.



## 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 your app.

[Tutorial: Map custom DNS name to your app](#)

Or, check out other resources:

[Configure PHP app](#)

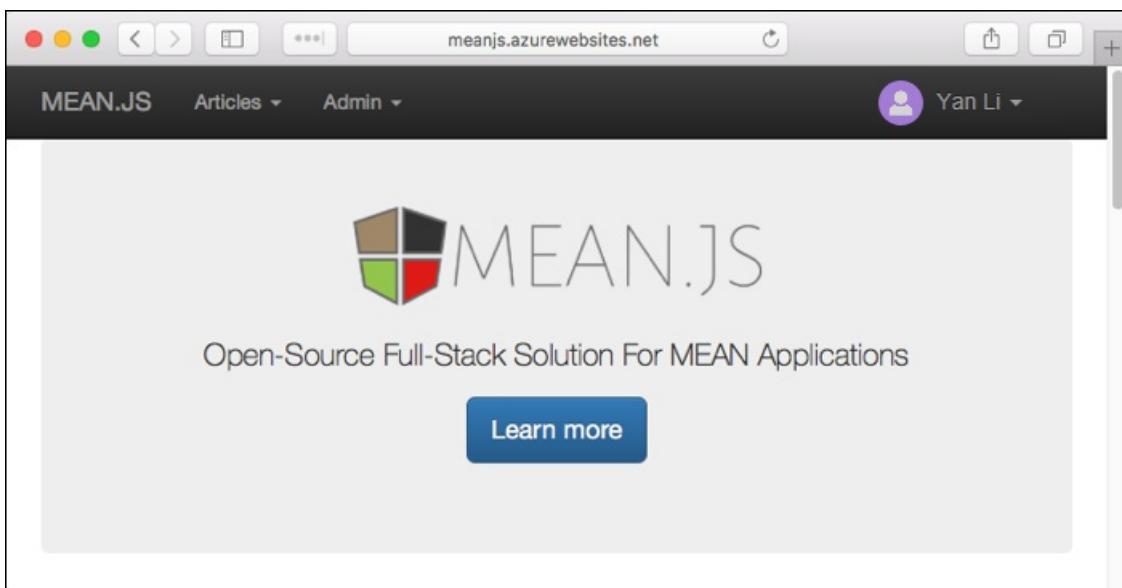
# Build a Node.js and MongoDB app in Azure App Service on Linux

2/21/2020 • 15 minutes to read • [Edit Online](#)

## NOTE

This article deploys an app to App Service on Linux. To deploy to App Service on Windows, see [Build a Node.js and MongoDB app in Azure](#).

[App Service on Linux](#) provides a highly scalable, self-patching web hosting service using the Linux operating system. This tutorial shows how to create a Node.js app, 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. For simplicity, the sample application uses the [MEAN.js web framework](#).



In this tutorial, you learn how to:

- Create a database using Azure Cosmos DB's API for MongoDB
- 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:

1. [Install Git](#)
2. [Install Node.js v6.0 or above and NPM](#)
3. [Install Gulp.js \(required by MEAN.js\)](#)

## 4. Install and run MongoDB Community Edition

### Test local MongoDB

Open the terminal window and `cd` to the `bin` directory of your MongoDB installation. You can use this terminal window to run all the commands in this tutorial.

Run `mongo` in the terminal to connect to your local MongoDB server.

```
mongo
```

If your connection is successful, then your MongoDB database is already running. If not, make sure that your local MongoDB database is started by following the steps at [Install MongoDB Community Edition](#). Often, MongoDB is installed, but you still need to start it by running `mongod`.

When you're done testing your MongoDB database, type `Ctrl+C` in the terminal.

### 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/meanjs.git
```

This sample repository contains a copy of the [MEAN.js repository](#). It is modified to run on App Service (for more information, see the MEAN.js repository [README file](#)).

#### Run the application

Run the following commands to install the required packages and start the application.

```
cd meanjs
npm install
npm start
```

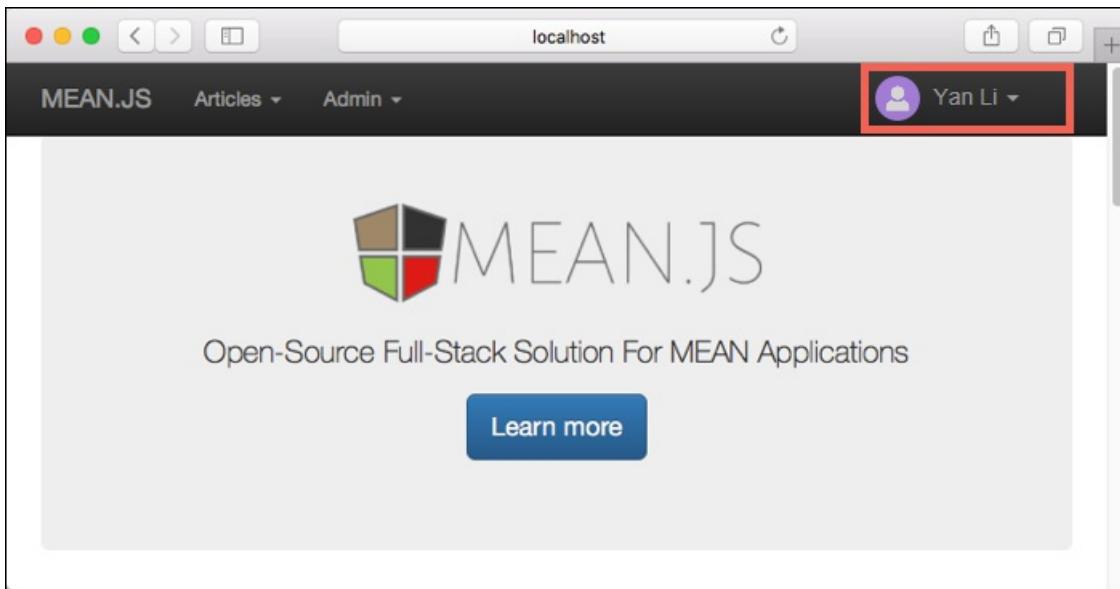
Ignore the config.domain warning. When the app is fully loaded, you see something similar to the following message:

```
-- 
MEAN.JS - Development Environment

Environment:      development
Server:          http://0.0.0.0:3000
Database:        mongodb://localhost/mean-dev
App version:     0.5.0
MEAN.JS version: 0.5.0
--
```

Navigate to `http://localhost:3000` in a browser. Click **Sign Up** in the top menu and create a test user.

The MEAN.js sample application stores user data in the database. If you are successful at creating a user and signing in, then your app is writing data to the local MongoDB database.



Select **Admin > Manage Articles** to add some articles.

To stop Node.js 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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.

## Create production MongoDB

In this step, you create a database account using Azure Cosmos DB's API for MongoDB. When your app is deployed to Azure, it uses this cloud database.

## Create a resource group

A [resource group](#) is a logical container into which Azure resources like 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 a Cosmos DB account

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:

```
{
  "consistencyPolicy": {
    "defaultConsistencyLevel": "Session",
    "maxIntervalInSeconds": 5,
    "maxStalenessPrefix": 100
  },
  "databaseAccountOfferType": "Standard",
  "documentEndpoint": "https://<cosmosdb-name>.documents.azure.com:443/",
  "failoverPolicies": ...
  < Output truncated for readability >
}
```

## Connect app to production configured with Azure Cosmos DB's API for MongoDB

In this step, you connect your MEAN.js 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 list-keys` command to retrieve the primary key.

```
az cosmosdb list-keys --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 Node.js application

In your local MEAN.js repository, in the `config/env` folder, create a file named `local-production.js`. `.gitignore` is configured to keep this file out of the repository.

Copy the following code into it. Be sure to replace the two `<cosmosdb-name>` placeholders with your Cosmos DB database name, and replace the `<primary-master-key>` placeholder with the key you copied in the previous step.

```
module.exports = {  
  db: {  
    uri: 'mongodb://<cosmosdb-name>:<primary-master-key>@<cosmosdb-name>.documents.azure.com:10250/mean?  
    ssl=true&sslverifycertificate=false'  
  }  
};
```

The `ssl=true` option is required because [Cosmos DB requires SSL](#).

Save your changes.

### Test the application in production mode

In a local terminal window, run the following command to minify and bundle scripts for the production environment. This process generates the files needed by the production environment.

```
gulp prod
```

In a local terminal window, run the following command to use the connection string you configured in `config/env/local-production.js`. Ignore the certificate error and the config.domain warning.

```
NODE_ENV=production node server.js
```

`NODE_ENV=production` sets the environment variable that tells Node.js to run in the production environment. `node server.js` starts the Node.js server with `server.js` in your repository root. This is how your Node.js application is loaded in Azure.

When the app is loaded, check to make sure that it's running in the production environment:

```
--  
MEAN.JS  
  
Environment: production  
Server: http://0.0.0.0:8443  
Database: mongodb://<cosmosdb-name>:<primary-master-key>@<cosmosdb-name>.documents.azure.com:10250/mean?ssl=true&sslverifycertificate=false  
App version: 0.5.0  
MEAN.JS version: 0.5.0
```

Navigate to `http://localhost:8443` in a browser. Click **Sign Up** in the top menu and create a test user. If you are successful creating a user and signing in, then your app is writing data to 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 Node.js application to Azure 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 in the resource group with the `az appservice plan create` command.

The following example creates an App Service plan named `myAppServicePlan` in the **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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|6.9`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash  
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime  
"NODE|6.9" --deployment-local-git  
# PowerShell  
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime  
"NODE|6.9" --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.

## Configure an environment variable

By default, the MEAN.js project keeps `config/env/local-production.js` out of the Git repository. So for your Azure app, you use app settings to define your MongoDB connection string.

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 `<primary-master-key>` placeholders.

```
az webapp config appsettings set --name <app-name> --resource-group myResourceGroup --settings  
MONGODB_URI="mongodb://<cosmosdb-name>:<primary-master-key>@<cosmosdb-name>.documents.azure.com:10250/mean?  
ssl=true"
```

In Node.js code, you [access this app setting](#) with `process.env.MONGODB_URI`, just like you would access any environment variable.

In your local MEAN.js repository, open `config/env/production.js` (not `config/env/local-production.js`), which has production-environment specific configuration. The default MEAN.js app is already configured to use the `MONGODB_URI` environment variable that you created.

```
db: {  
  uri: ... || process.env.MONGODB_URI || ...,  
  ...  
},
```

## Push to Azure from Git

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>
```

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 master
```

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

```
Counting objects: 5, done.  
Delta compression using up to 4 threads.  
Compressing objects: 100% (5/5), done.  
Writing objects: 100% (5/5), 489 bytes | 0 bytes/s, done.  
Total 5 (delta 3), reused 0 (delta 0)  
remote: Updating branch 'master'.  
remote: Updating submodules.  
remote: Preparing deployment for commit id '6c7c716ee'.  
remote: Running custom deployment command...  
remote: Running deployment command...  
remote: Handling node.js deployment.  
.  
.  
.  
remote: Deployment successful.  
To https://<app-name>.scm.azurewebsites.net/<app-name>.git  
 * [new branch] master -> master
```

You may notice that the deployment process runs [Gulp](#) after `npm install`. App Service does not run Gulp or Grunt tasks during deployment, so this sample repository has two 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 `gulp prod` after `npm install` and `bower install`.

You can use this approach to add any step to your Git-based deployment. If you restart your Azure app at any point, App Service doesn't rerun these automation tasks. For more information, see [Run Grunt/Bower/Gulp](#).

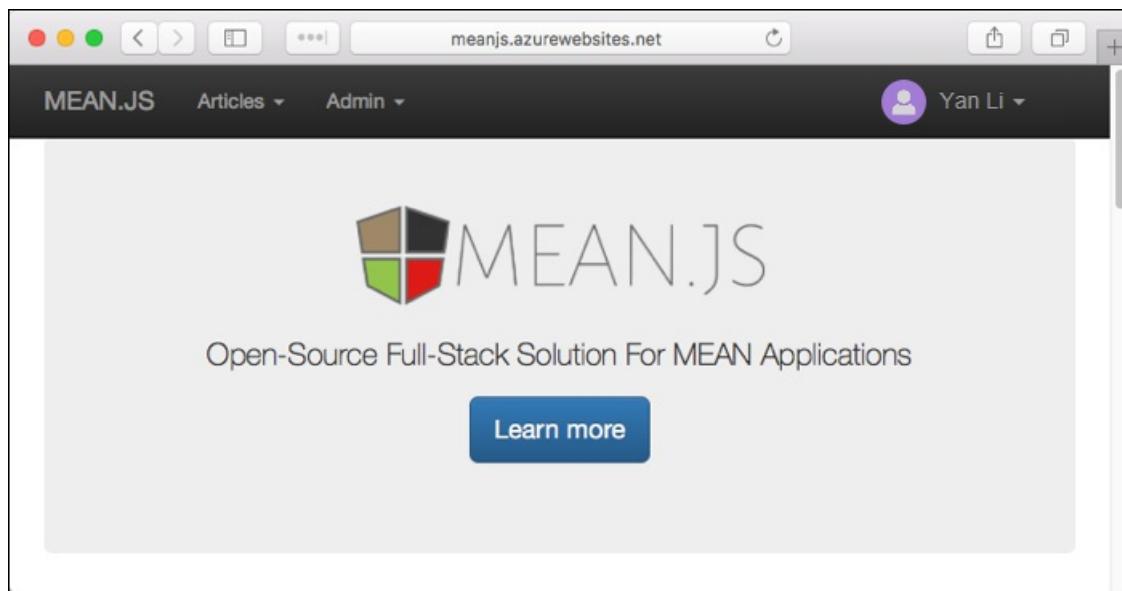
## Browse to the Azure app

Browse to the deployed app using your web browser.

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

Click **Sign Up** in the top menu and create a dummy user.

If you are successful and the app automatically signs in to the created user, then your MEAN.js app in Azure has connectivity to the Azure Cosmos DB's API for MongoDB.



Select **Admin > Manage Articles** to add some articles.

**Congratulations!** You're running a data-driven Node.js app in Azure App Service on Linux.

## Update data model and redeploy

In this step, you change the `article` data model and publish your change to Azure.

### Update the data model

In your local MEAN.js repository, open `modules/articles/server/models/article.server.model.js`.

In `ArticleSchema`, add a `String` type called `comment`. When you're done, your schema code should look like this:

```
let ArticleSchema = new Schema({
  ...,
  user: {
    type: Schema.ObjectId,
    ref: 'User'
  },
  comment: {
    type: String,
    default: '',
    trim: true
  }
});
```

## Update the articles code

Update the rest of your `articles` code to use `comment`.

There are five files you need to modify: the server controller and the four client views.

Open `modules/articles/server/controllers/articles.server.controller.js`.

In the `update` function, add an assignment for `article.comment`. The following code shows the completed `update` function:

```
exports.update = function (req, res) {
  let article = req.article;

  article.title = req.body.title;
  article.content = req.body.content;
  article.comment = req.body.comment;

  ...
};
```

Open `modules/articles/client/views/view-article.client.view.html`.

Just above the closing `</section>` tag, add the following line to display `comment` along with the rest of the article data:

```
<p class="lead" ng-bind="vm.article.comment"></p>
```

Open `modules/articles/client/views/list-articles.client.view.html`.

Just above the closing `</a>` tag, add the following line to display `comment` along with the rest of the article data:

```
<p class="list-group-item-text" ng-bind="article.comment"></p>
```

Open `modules/articles/client/views/admin/list-articles.client.view.html`.

Inside the `<div class="list-group">` element and just above the closing `</a>` tag, add the following line to display `comment` along with the rest of the article data:

```
<p class="list-group-item-text" data-ng-bind="article.comment"></p>
```

Open `modules/articles/client/views/admin/form-article.client.view.html`.

Find the `<div class="form-group">` element that contains the submit button, which looks like this:

```
<div class="form-group">
  <button type="submit" class="btn btn-default">{{vm.article._id ? 'Update' : 'Create'}}</button>
</div>
```

Just above this tag, add another `<div class="form-group">` element that lets people edit the `comment` field. Your new element should look like this:

```
<div class="form-group">
  <label class="control-label" for="comment">Comment</label>
  <textarea name="comment" data-ng-model="vm.article.comment" id="comment" class="form-control" cols="30"
rows="10" placeholder="Comment"></textarea>
</div>
```

## Test your changes locally

Save all your changes.

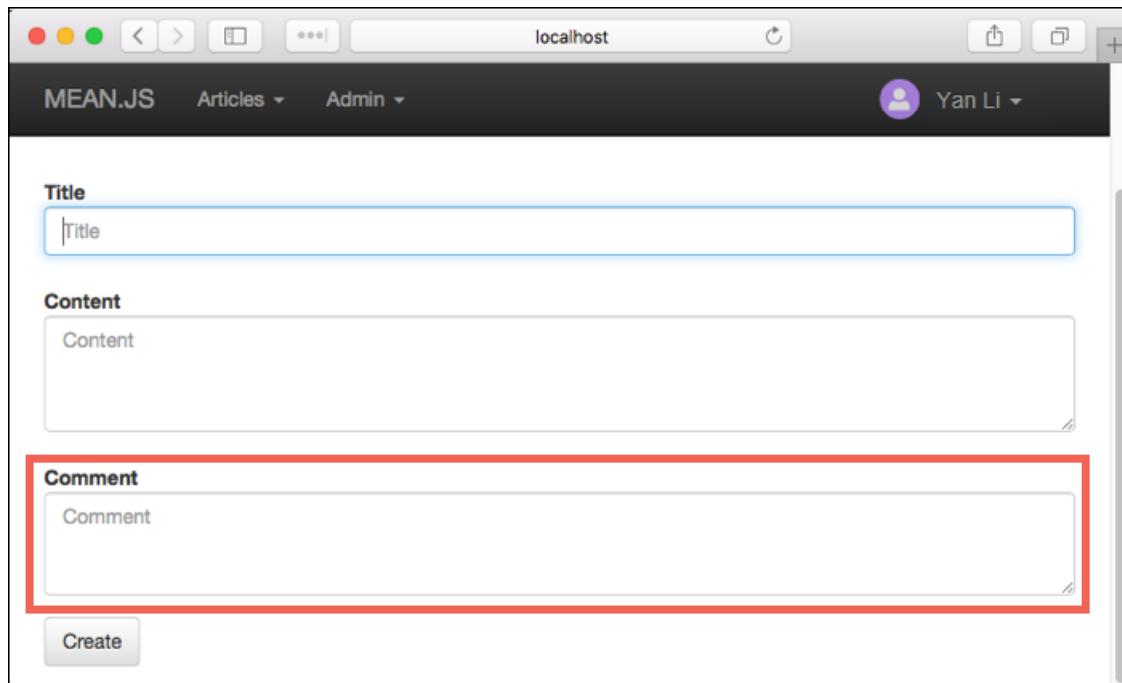
In the local terminal window, test your changes in production mode again.

```
gulp prod
NODE_ENV=production node server.js
```

Navigate to `http://localhost:8443` in a browser and make sure that you're signed in.

Select **Admin > Manage Articles**, then add an article by selecting the + button.

You see the new `Comment` textbox now.



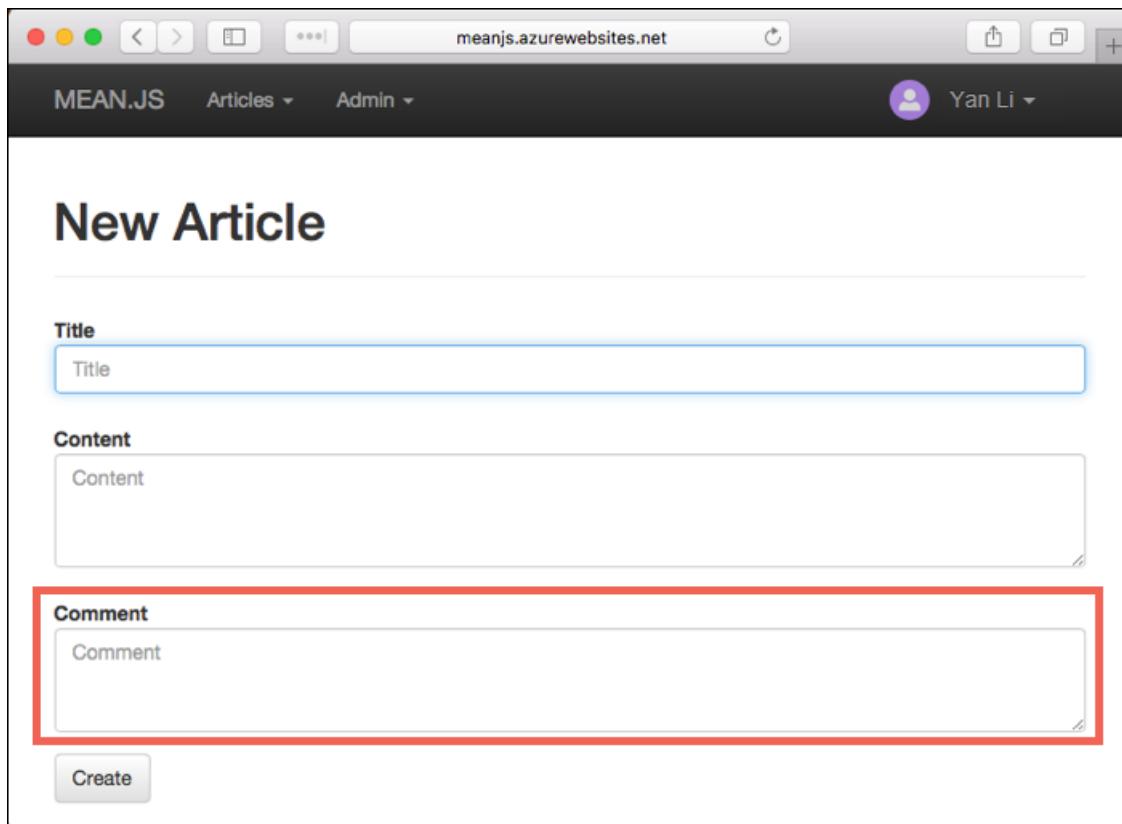
In the terminal, stop Node.js by typing `Ctrl+C`.

## Publish changes to Azure

Commit your changes in Git, then push the code changes to Azure.

```
git commit -am "added article comment"
git push azure master
```

Once the `git push` is complete, navigate to your Azure app and try out the new functionality.



The screenshot shows a web browser window for 'meanjs.azurewebsites.net'. The page title is 'MEAN.JS'. The top navigation bar includes 'Articles' and 'Admin' dropdowns, and a user profile for 'Yan Li'. Below the navigation is a section titled 'New Article'. It contains three input fields: 'Title' (placeholder 'Title'), 'Content' (placeholder 'Content'), and 'Comment' (placeholder 'Comment'). A red rectangular box highlights the 'Comment' input field. At the bottom is a 'Create' button.

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

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

### 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 'Web', 'SQL', and 'File'. The 'Web' icon is highlighted with a red box. The main area displays a table with one item. The columns are 'NAME', 'STATUS', 'APP TYPE', 'APP SERVICE PLAN', and 'LOCATION'. The single row shows 'meanjs' as the name, 'Running' as the status, 'Web app' as the type, 'myAppServicePlan' as the service plan, and 'West Europe' as the location.

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 'meanjs' Overview page. The left sidebar has tabs for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Diagnose and solve problems', 'Deployment', 'Quickstart', and 'Deployment credentials'. The 'Overview' tab is selected and highlighted with a blue background. The main content area shows the following details:

- Resource group (change)**: myResourceGroup
- Status**: Running
- Location**: West Europe
- Subscription name (change)**: [redacted]
- Subscription ID**: [redacted]
- URL**: http://meanjs.azurewebsites.net
- App Service plan/pricing tier**: myAppServicePlan (Free)
- Git/Deployment username**: [redacted]
- Git clone url**: [redacted]
- FTP hostname**: ftp://waws-prod-am2-025.ftp.azurewebsit...

## 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 database using Azure Cosmos DB's API for MongoDB
- Connect a Node.js app to a 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](#)

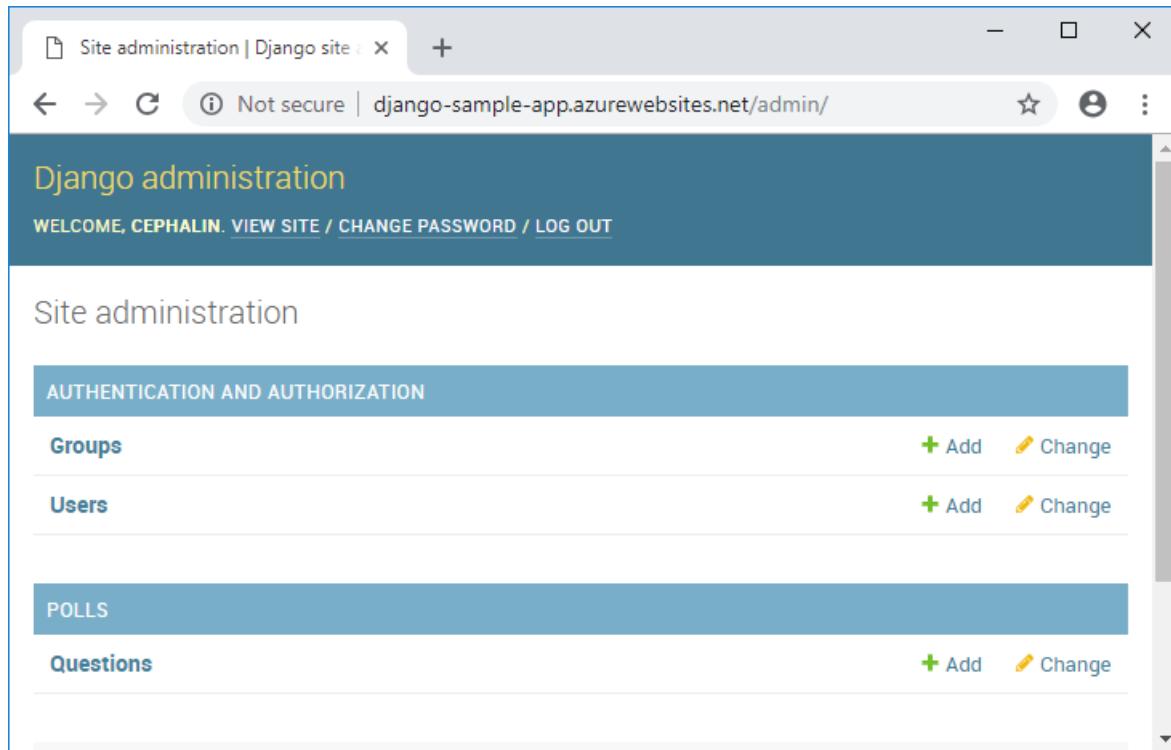
Or, check out other resources:

[Configure Node.js app](#)

# Tutorial: Run a Python (Django) web app with PostgreSQL in Azure App Service

2/21/2020 • 15 minutes to read • [Edit Online](#)

Azure App Service provides a highly scalable, self-patching web hosting service. This tutorial shows how to connect a data-driven Python Django web app to an Azure Database for PostgreSQL database, and deploy and run the app on Azure App Service.



In this tutorial, you learn how to:

- Create an Azure Database for PostgreSQL database and connect a web app to it
- Deploy the web app to Azure App Service
- View diagnostic logs
- Manage the web app in the Azure portal

You can follow the steps in this article on macOS, Linux, or Windows. The steps are similar in most cases, although differences aren't detailed in this tutorial. Most of the examples below use a `bash` terminal window on Linux.

## Prerequisites

Before you start this tutorial:

- If you don't have an [Azure subscription](#), create a [free account](#) before you begin.
- Install [Git](#).
- Install [Python 3](#).
- Install and run [PostgreSQL](#).

## Test PostgreSQL installation and create a database

First, connect to your local PostgreSQL server and create a database:

In a local terminal window, run `psql` to connect to your local PostgreSQL server as the built-in `postgres` user.

```
sudo su - postgres  
psql
```

or

```
psql -U postgres
```

If your connection is successful, your PostgreSQL database is running. If not, make sure that your local PostgreSQL database is started by following the instructions for your operating system at [Downloads - PostgreSQL Core Distribution](#).

Create a new database called `pollsdb`, and set up a database user named `manager` with password `supersecretpass`:

```
CREATE DATABASE pollsdb;  
CREATE USER manager WITH PASSWORD 'supersecretpass';  
GRANT ALL PRIVILEGES ON DATABASE pollsdb TO manager;
```

Type `\q` to exit the PostgreSQL client.

## Create and run the local Python app

Next, set up and run the sample Python Django web app.

The [djangoapp](#) sample repository contains the data-driven [Django](#) polls app you get by following [Writing your first Django app](#) in the Django documentation.

### Clone the sample app

In a terminal window, run the following commands to clone the sample app repository, and change to the new working directory:

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

### Configure the Python virtual environment

Create and activate a Python virtual environment to run your app.

```
python3 -m venv venv  
source venv/bin/activate
```

or

```
py -3 -m venv venv  
venv\scripts\activate
```

In the `venv` environment, run `env.sh` or `env.ps1` to set the environment variables that `azuresite/settings.py` will use for the database connection settings.

```
source ./env.sh
```

or

```
.\env.ps1
```

Install the required packages from *requirements.txt*, run [Django migrations](#), and [create an admin user](#):

```
pip install -r requirements.txt
python manage.py migrate
python manage.py createsuperuser
```

## Run the web app

After you create the admin user, run the Django server.

```
python manage.py runserver
```

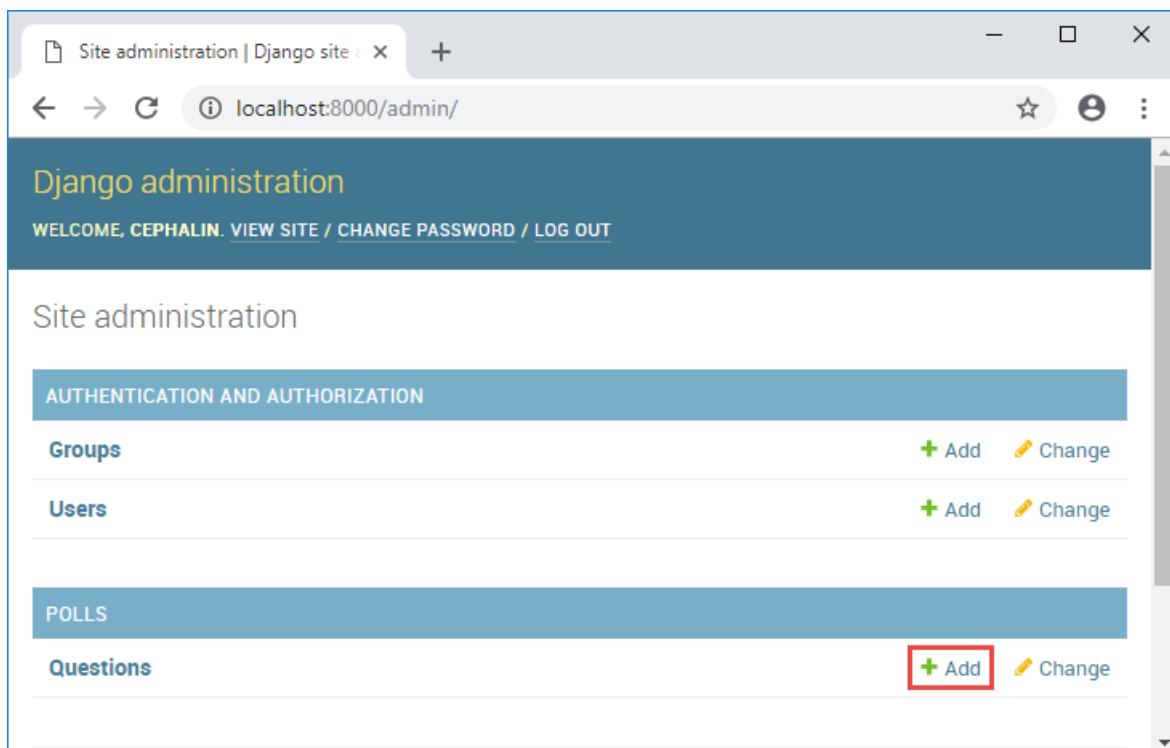
When the Django web app is fully loaded, it returns something like the following message:

```
Performing system checks...

System check identified no issues (0 silenced).
December 13, 2019 - 10:54:59
Django version 2.1.2, using settings 'azuresite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Go to <http://localhost:8000> in a browser. You should see the message **No polls are available**.

Go to <http://localhost:8000/admin> and sign in using the admin user you created in the last step. Select **Add** next to **Questions**, and create a poll question with some choices.



Go to <http://localhost:8000> again to see the poll question and answer the question. The local Django sample

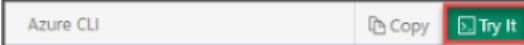
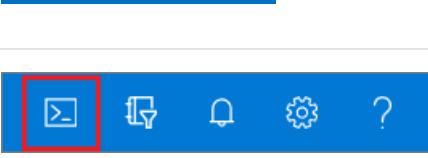
application writes and stores user data to the local PostgreSQL database.

To stop the Django server, type **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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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.

Most of the remaining steps in this article use Azure CLI commands in the Azure Cloud Shell.

## Create and connect to Azure Database for PostgreSQL

In this section, you create an Azure Database for PostgreSQL server and database, and connect your web app to it. When you deploy your web app to Azure App Service, the app uses this cloud database.

### Create a resource group

You can create a new resource group for your Azure Database for PostgreSQL server, or use an existing resource group.

A [resource group](#) is a logical container into which Azure resources like 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 Database for PostgreSQL server

You create a PostgreSQL server with the [az postgres server create](#) command in the Cloud Shell.

#### NOTE

Before you create an Azure Database for PostgreSQL server, check which [compute generation](#) is available in your region. If your region doesn't support Gen4 hardware, change `--sku-name` in the following command line to a value that's supported in your region, such as Gen5.

In the following command, replace `<postgresql-name>` with a unique server name. The server name is part of your PostgreSQL endpoint <https://<postgresql-name>.postgres.database.azure.com>, so the name needs to be unique across all servers in Azure.

Replace `<resourcegroup-name>` and `<region>` with the name and region of the resource group you want to use. For `<admin-username>` and `<admin-password>`, create user credentials for the database administrator account. Remember the `<admin-username>` and `<admin-password>` to use later for signing in to the PostgreSQL server and databases.

```
az postgres server create --resource-group <resourcegroup-name> --name <postgresql-name> --location "<region>"  
--admin-user <admin-username> --admin-password <admin-password> --sku-name B_Gen5_1
```

When the Azure Database for PostgreSQL server is created, the Azure CLI returns JSON code like the following example:

```
{  
  "administratorLogin": "myusername",  
  "earliestRestoreDate": "2020-01-22T19:02:15.727000+00:00",  
  "fullyQualifiedDomainName": "myservername.postgres.database.azure.com",  
  "id": "/subscriptions/00000000-0000-0000-0000-  
0000000000/resourceGroups/myresourcegroup/providers/Microsoft.DBforPostgreSQL/servers/myservername",  
  "location": "westeurope",  
  "masterServerId": "",  
  "name": "myservername",  
  "replicaCapacity": 5,  
  "replicationRole": "None",  
  "resourceGroup": "myresourcegroup",  
  "sku": {  
    "capacity": 1,  
    "family": "Gen5",  
    "name": "B_Gen5_1",  
    "size": null,  
    "tier": "Basic"  
  },  
  < JSON data removed for brevity. >  
}
```

### Create firewall rules for the Azure Database for PostgreSQL server

Run the [az postgres server firewall-rule create](#) command to allow access to the database from Azure resources.

Replace the `<postgresql-name>` and `<resourcegroup-name>` placeholders with your values.

```
az postgres server firewall-rule create --resource-group <resourcegroup-name> --server-name <postgresql-name>  
--start-ip-address=0.0.0.0 --end-ip-address=0.0.0.0 --name AllowAllAzureIPs
```

#### NOTE

The preceding setting allows network connections from all IP addresses within the Azure network. For production use, try to configure the most restrictive firewall rules possible by [allowing only the outbound IP addresses your app uses](#).

Run the `firewall-rule create` command again to allow access from your local computer. Replace `<your-ip-address>` with [your local IPv4 IP address](#). Replace the `<postgresql-name>` and `<resourcegroup-name>` placeholders with your own values.

```
az postgres server firewall-rule create --resource-group <resourcegroup-name> --server-name <postgresql-name> --start-ip-address=<your-ip-address> --end-ip-address=<your-ip-address> --name AllowLocalClient
```

#### Create and connect to the Azure Database for PostgreSQL database

Connect to your Azure Database for PostgreSQL server by running the following command. Use your own `<postgresql-name>` and `<admin-username>`, and sign in with the password you created.

```
psql -h <postgresql-name>.postgres.database.azure.com -U <admin-username>@<postgresql-name> postgres
```

Just as you did in your local PostgreSQL server, create a database and user in the Azure Database for PostgreSQL server:

```
CREATE DATABASE pollsdb;
CREATE USER manager WITH PASSWORD 'supersecretpass';
GRANT ALL PRIVILEGES ON DATABASE pollsdb TO manager;
```

#### NOTE

It's a best practice to create database users with restricted permissions for specific apps, instead of using the admin user. The `manager` user has full privileges to *only* the `pollsdb` database.

Type `\q` to exit the PostgreSQL client.

#### Test app connectivity to the Azure PostgreSQL database

Edit your local `env.sh` or `env.ps1` file to point to to your cloud PostgreSQL database, by replacing `<postgresql-name>` with your Azure Database for PostgreSQL server name.

```
export DBHOST=<postgresql-name>.postgres.database.azure.com"
export DBUSER="manager@<postgresql-name>"
export DBNAME="pollsdb"
export DBPASS="supersecretpass"
```

or

```
$Env:DBHOST = "<postgresql-name>.postgres.database.azure.com"
$Env:DBUSER = "manager@<postgresql-name>"
$Env:DBNAME = "pollsdb"
$Env:DBPASS = "supersecretpass"
```

In the `venv` environment in your local terminal window, run the edited `env.sh` or `env.ps1`.

```
source ./env.sh
```

or

```
.\env.ps1
```

Run Django migration to the Azure database, and create an admin user.

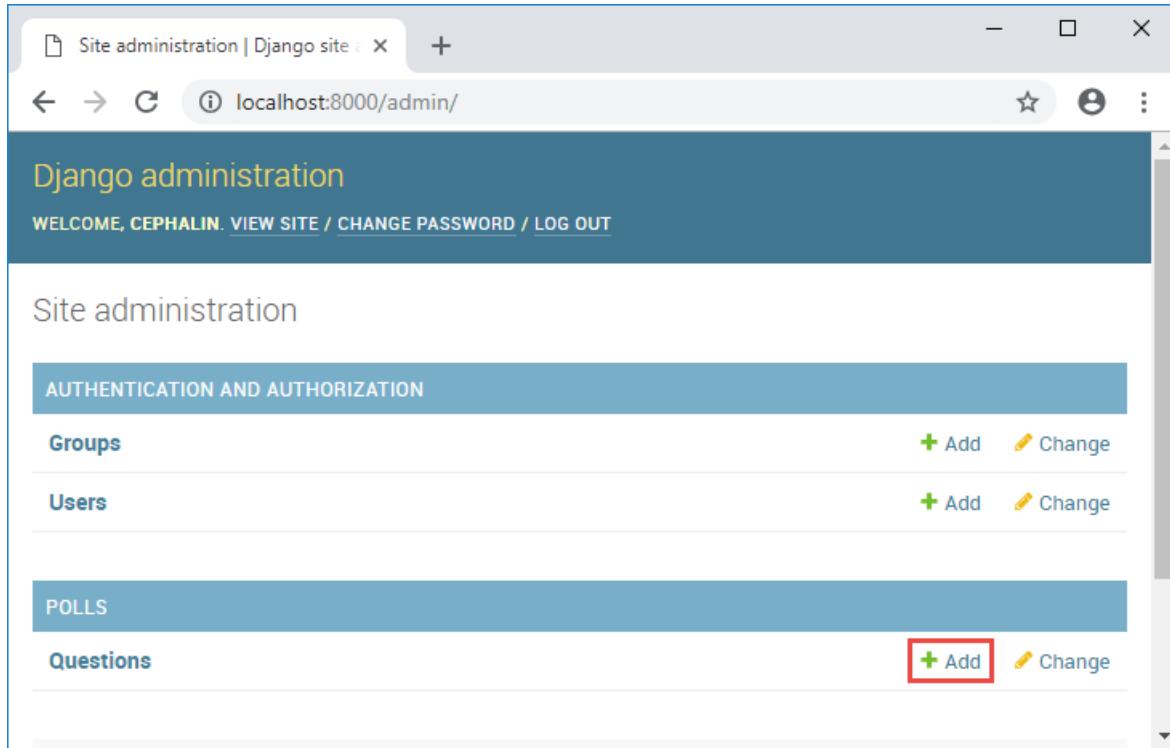
```
python manage.py migrate  
python manage.py createsuperuser
```

Once the admin user is created, run the Django server.

```
python manage.py runserver
```

In a browser, go to <http://localhost:8000>, and you should see the message **No polls are available** again.

Go to <http://localhost:8000/admin>, sign in using the admin user you created, and create a poll question like before.



Go to <http://localhost:8000> again, and see the poll question displayed. Your app is now writing data to the Azure Database for PostgreSQL database.

To stop the Django server, type Ctrl+C in the terminal.

## Deploy the web app to Azure App Service

In this step, you deploy the Azure Database for PostgreSQL database-connected Python app to Azure App Service.

### Configure repository

Because this tutorial uses a Django sample, you need to change and add some settings in your `djangoapp/azuresite/settings.py` file to work with Azure App Service.

1. Django validates the `HTTP_HOST` header in incoming requests. For your Django web app to work in App Service, you need to add the fully qualified domain name of the app to the allowed hosts.

Edit `azuresite/settings.py` to change the `ALLOWED_HOSTS` line as follows:

```
ALLOWED_HOSTS = [os.environ['WEBSITE_SITE_NAME'] + '.azurewebsites.net', '127.0.0.1'] if  
'WEBSITE_SITE_NAME' in os.environ else []
```

2. Django doesn't support [serving static files in production](#). For this tutorial, you use [WhiteNoise](#) to enable serving the files. The [WhiteNoise](#) package was already installed with `requirements.txt`.

To configure Django to use WhiteNoise, in `azuresite/settings.py`, find the `MIDDLEWARE` setting, and add `whitenoise.middleware.WhiteNoiseMiddleware` to the list, just after the `django.middleware.security.SecurityMiddleware` line. Your `MIDDLEWARE` setting should look like this:

```
MIDDLEWARE = [  
    'django.middleware.security.SecurityMiddleware',  
    'whitenoise.middleware.WhiteNoiseMiddleware',  
    ...  
]
```

3. At the end of `azuresite/settings.py`, add the following lines:

```
STATICFILES_STORAGE = 'whitenoise.storage.CompressedManifestStaticFilesStorage'  
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
```

For more information about configuring WhiteNoise, see the [WhiteNoise documentation](#).

#### IMPORTANT

The database settings section already follows the security best practice of using environment variables. For complete deployment recommendations, see [Django Documentation: deployment checklist](#).

Commit your changes into your fork of the `djangoapp` repository:

```
git commit -am "configure for 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 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 **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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 `PYTHON|3.7`. To see all supported runtimes, run `az webapp list-runtimes --linux`.

```
# Bash
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PYTHON|3.7" --deployment-local-git
# PowerShell
az --% webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --runtime
"PYTHON|3.7" --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 environment variables

Earlier in the tutorial, you defined environment variables to connect to your PostgreSQL database.

In Azure App Service, you set environment variables as *app settings*, by using the [az webapp config appsettings set](#) command.

In the Azure Cloud Shell, run the following command to specify the database connection details as app settings.

Replace `<app-name>`, `<resourcegroup-name>`, and `<postgresql-name>` with your own values.

```
az webapp config appsettings set --name <app-name> --resource-group <resourcegroup-name> --settings DBHOST="<postgresql-name>.postgres.database.azure.com" DBUSER="manager@<postgresql-name>" DBPASS="supersecretpass" DBNAME="pollsdb"
```

For information on how your code accesses these app settings, see [Access environment variables](#).

## Push to Azure from Git

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>
```

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 master
```

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

```
Counting objects: 60, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (51/51), done.  
Writing objects: 100% (60/60), 15.37 KiB | 749.00 KiB/s, done.  
Total 60 (delta 9), reused 0 (delta 0)  
remote: Deploy Async  
remote: Updating branch 'master'.  
remote: Updating submodules.  
remote: Preparing deployment for commit id '06f3f7c0cb'.  
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: Done in 100 sec(s).  
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/06f3f7c0cb52ce3b4aff85c2b5099fbacb65ab94/log'  
To https://<app-name>.scm.azurewebsites.net/<app-name>.git  
 * [new branch] master -> master
```

The App Service deployment server sees *requirements.txt* in the repository root and runs Python package management automatically after `git push`.

### Browse to the Azure app

Browse to the deployed app with URL `http://<app-name>.azurewebsites.net`. It takes some time to start, because the container must be downloaded and run when the app is requested for the first time. If the page times out or displays an error message, wait a few minutes and refresh the page.

You should see the poll questions that you created earlier.

App Service detects a Django project in your repository by looking for a *wsgi.py* file in each subdirectory, which `manage.py startproject` creates by default. When App Service finds the file, it loads the Django web app. For more information on how App Service loads Python apps, see [Configure built-in Python image](#).

Go to `http://<app-name>.azurewebsites.net/admin` and sign in using the admin user you created. If you like, create some more poll questions.

The screenshot shows the Django administration interface. At the top, there's a header bar with the title "Site administration | Django site" and a "Not secure" warning. Below the header, the main title is "Django administration" and the sub-header is "WELCOME, CEPHALIN. VIEW SITE / CHANGE PASSWORD / LOG OUT". The main content area is titled "Site administration" and contains two sections: "AUTHENTICATION AND AUTHORIZATION" and "POLLING".

- AUTHENTICATION AND AUTHORIZATION:** Contains "Groups" and "Users" sections. Each section has a "Add" button (green plus sign) and a "Change" button (pencil icon).
- POLLING:** Contains "Questions" section. It also has a "Add" button (green plus sign) and a "Change" button (pencil icon).

**Congratulations!** You're running a Python (Django) web app in Azure App Service for Linux.

## Stream diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

### 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 app in the Azure portal

In the [Azure portal](#), search for and select the app you created.

My Dashboard

Auto refresh : Off

Services

No results were found.

Resources

postgres-app App Service

Marketplace

No results were found.

Documentation

Tutorial: Linux Ruby app with Postgres - Azure App Service ...

Resource Groups

No results were found.

Searching all subscriptions. [Change](#)

Last updated: a minute ago

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, restart, and delete. The tabs on the left side of the page show the different configuration pages you can open.

postgres-app

App Service

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Deployment

Quickstart

Deployment slots

Deployment Center

Settings

Configuration

Authentication / Authorization

Application Insights

Resource group (change)  
postgres

Status  
Running

Location  
West US 2

Subscription (change)  
A Subscription

Subscription ID  
12345678-1234-1234-1234-123456781234

Tags (change)  
Click here to add tags

Browse Stop Swap Restart Delete Get publish profile Reset publish profile

URL  
<https://postgres-app.azurewebsites.net>

App Service Plan  
[postgresAppServicePlan \(F1: Free\)](#)

Git/Deployment username  
azureuser10

Git clone url  
<https://azureuser10@postgres-app.scm.azurewebsites.net:443/p...>

FTP hostname  
<ftp://waws-prod-mwh-007.ftp.azurewebsites.windows.net>

Diagnose and solve problems

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

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.

## 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

Go to the next tutorial to learn how to map a custom DNS name to your app:

[Tutorial: Map custom DNS name to your app](#)

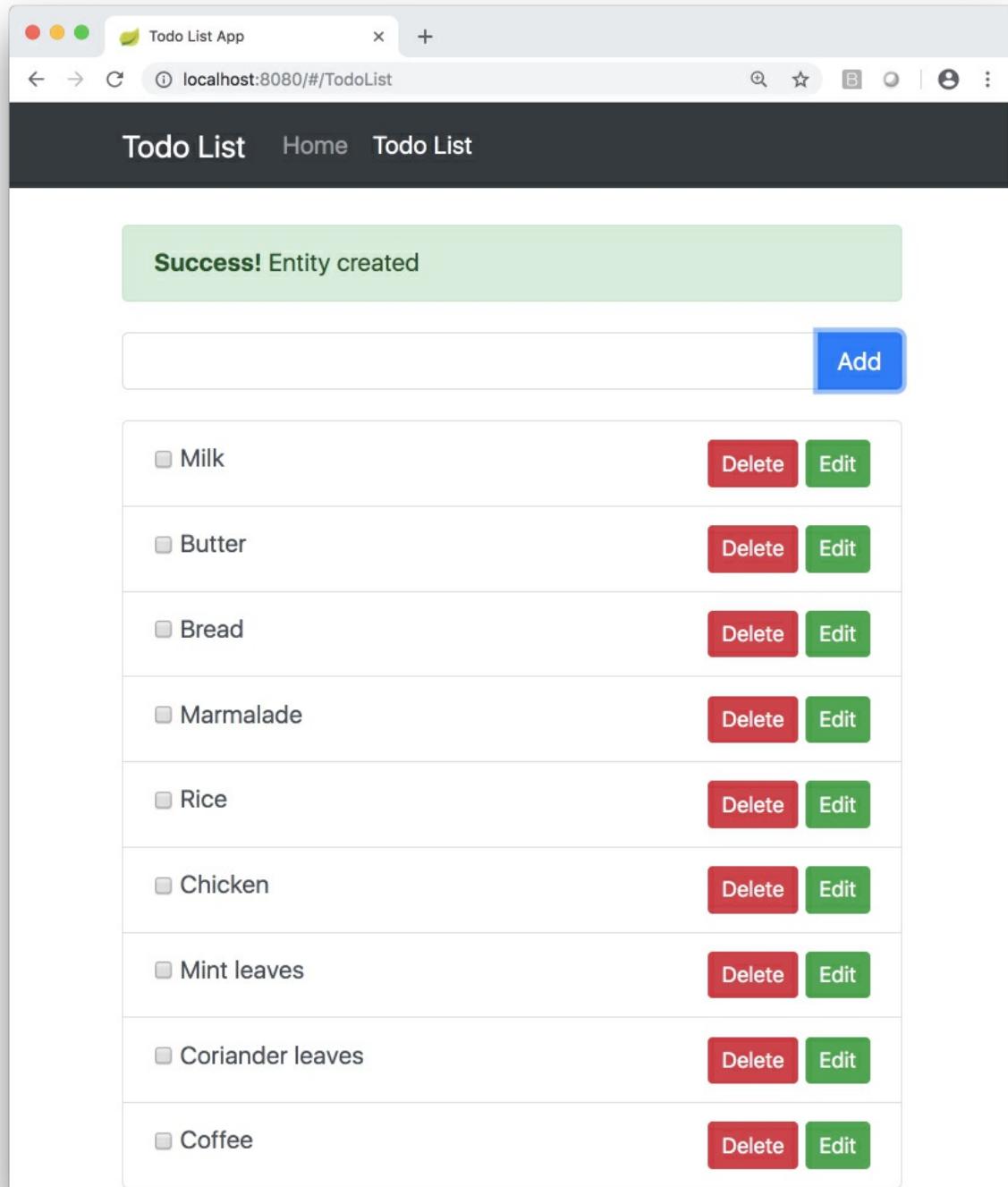
Or check out other resources:

[Configure Python app](#)

# Tutorial: Build a Java Spring Boot web app with Azure App Service on Linux and Azure Cosmos DB

2/17/2020 • 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 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 list-keys -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.

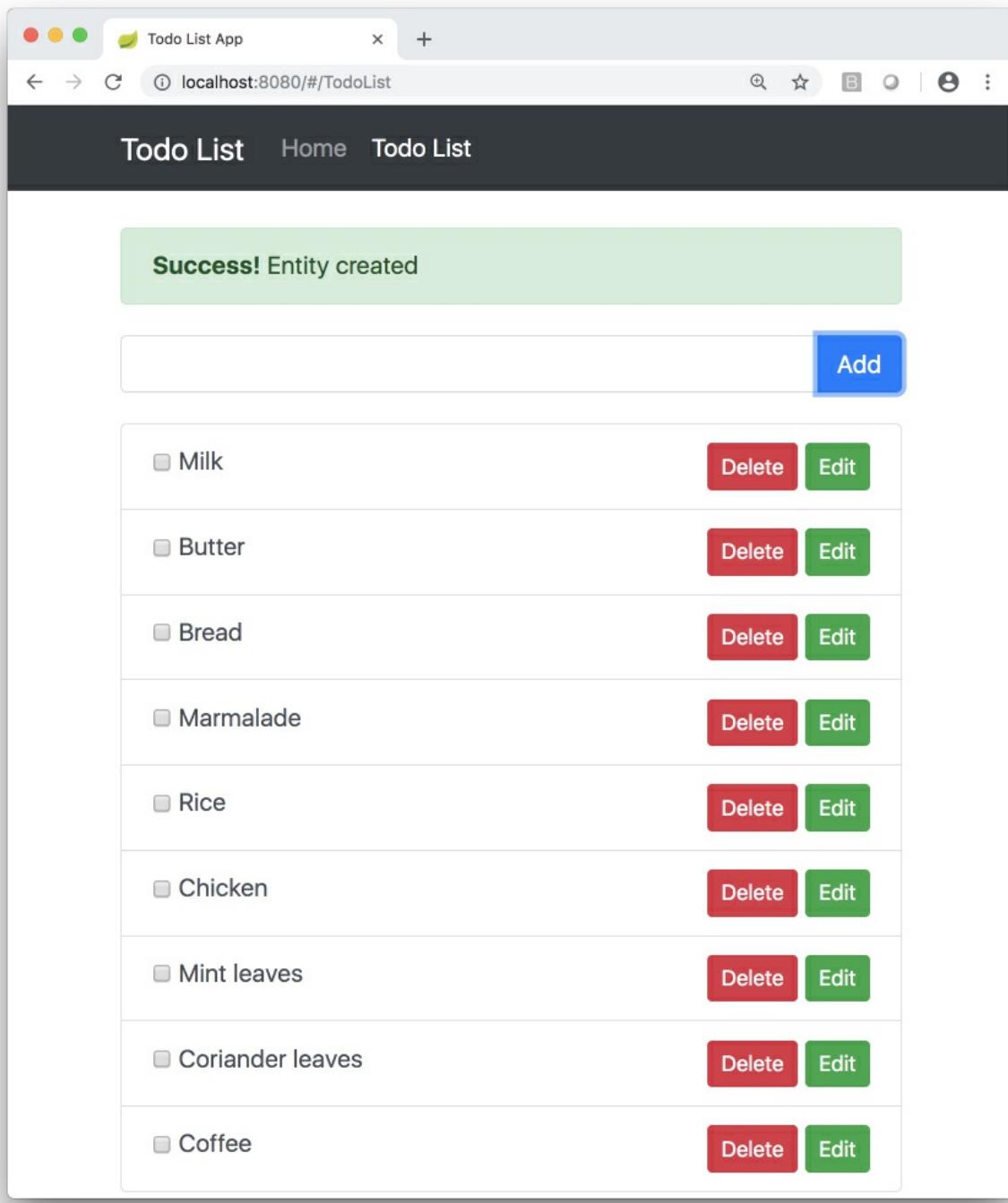
```
mvn package spring-boot:run
```

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>1.8.0</version>
        <configuration>
            <schemaVersion>v2</schemaVersion>

            <!-- Web App information -->
            <resourceGroup>${RESOURCEGROUP_NAME}</resourceGroup>
            <appName>${WEBAPP_NAME}</appName>
            <region>${REGION}</region>

            <!-- Java Runtime Stack for Web App on Linux-->
            <runtime>
                <os>linux</os>
                <javaVersion>jre8</javaVersion>
                <webContainer>jre8</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 `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:1.8.0:deploy (default-cli) @ spring-todo-app ---
[INFO] Target Web App doesn't exist. Creating a new one...
[INFO] Creating App Service Plan 'ServicePlanb6ba8178-5bbb-49e7'...
[INFO] Successfully created App Service Plan.
[INFO] Successfully created Web App.
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource to /home/test/e2e-java-experience-in-app-service-linux-part-2/initial/spring-todo-
app/target/azure-webapp/spring-todo-app-61bb5207-6fb8-44c4-8230-c1c9e4c099f7
[INFO] Trying to deploy artifact to spring-todo-app...
[INFO] Renaming /home/test/e2e-java-experience-in-app-service-linux-part-2/initial/spring-todo-
app/target/azure-webapp/spring-todo-app-61bb5207-6fb8-44c4-8230-c1c9e4c099f7/spring-todo-app-2.0-SNAPSHOT.jar
to app.jar
[INFO] Deploying the zip package spring-todo-app-61bb5207-6fb8-44c4-8230-c1c9e4c099f7718326714198381983.zip...
[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.

```
open 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 dark header with "Todo List" and "Home" buttons. Below the header is 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

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

#### 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>
```

## 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](#)

# Tutorial: Create a multi-container (preview) app in Web App for Containers

2/21/2020 • 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 like 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.

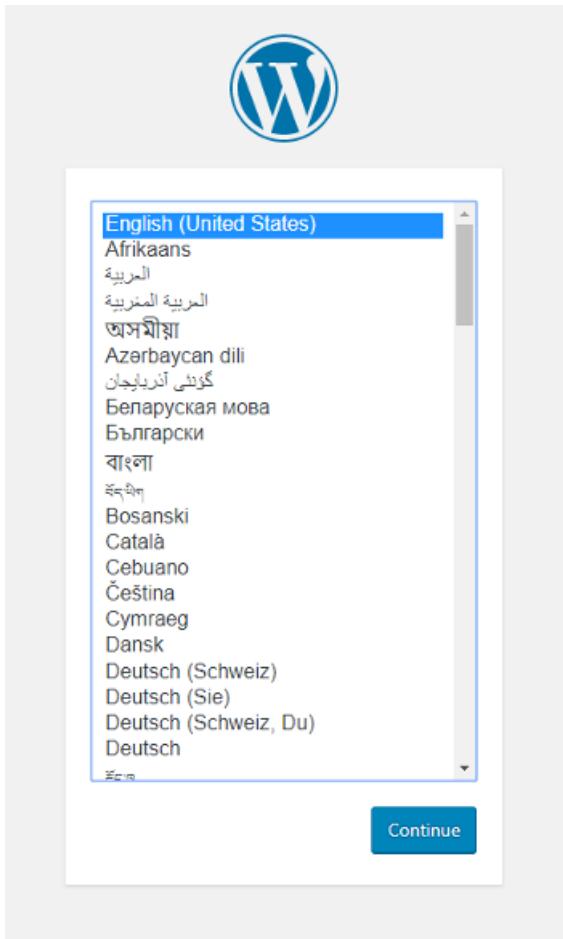
```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --multicontainer-config-type compose --multicontainer-config-file docker-compose-wordpress.yml
```

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_Gen4_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-  
00000000/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 SSL and other configurations

By default, SSL is used by Azure Database for MySQL. WordPress requires additional configuration to use 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: microsoft/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: microsoft/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.

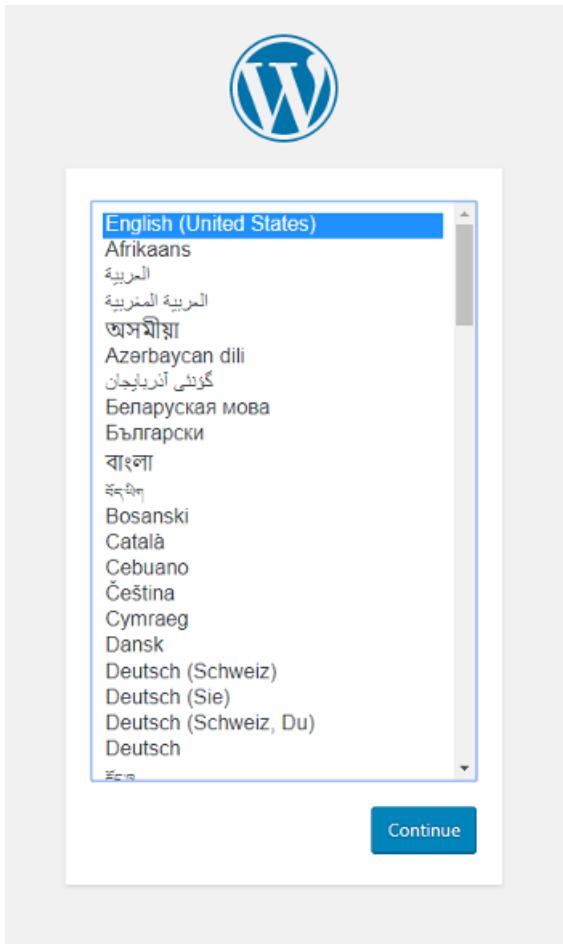
```
az webapp config container set --resource-group myResourceGroup --name <app-name> --multicontainer-config-type compose --multicontainer-config-file docker-compose-wordpress.yml
```

When the app has been reconfigured, Cloud Shell shows information similar to the following example:

```
[  
  {  
    "name": "DOCKER_CUSTOM_IMAGE_NAME",  
    "value":  
      "COMPOSE|dmVyc2lvbjogJzMycKCrNlcnZpY2VzOgogICB3b3JkcHJlc3M6CiAgICAgaW1hZ2U6IG1zYW5nYXB1L3dvcmlRwcmVzcwogICAgIHBvcnRzOgogICAgICAgLSAiODAwMDo4MCIKICAgICByZXN0YXJ0OiBhbHdheXM="  
  }  
]
```

## 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: microsoft/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|dmVyc2lvbjogJzMuMyCkCnNlcnPzY2Vz0ogICBteXNxDoKICAgICBpbWFnZToghXlzcWw6NS43CiAgICAgdm9sdW1lczokICAgI  
 CAgIC0gZGJfZGF0YTovdmFyL2xpYi9teXNx0AogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW500gogICAgICAgTV1TUxfUk9  
 PVF9Q0QNTV09SRD0gZXhhbXBsZX0hC3MKCiAgIhdvcmRwcmVzcoKICAgICBkZXBlbmRzX29u0gogICAgICAgLSBteXNx0AogICAgIGltYWd1o  
 iB3b3JkcHJlc3M6bGF0ZXN0CiAgICAgcG9ydHM6CiAgICAgICAtIC14MDAwOjgwIgogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgICAgIGVu  
 dmlyb25tZW500gogICAgICAgV09SRFB0RVNTX0RCX1BBU1NXT1JE0iBleGFtcGxlcGFzcwcp2b2x1bwVz0gogICAgZGJfZGF0YT0="  
 }  
 ]
```

## 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: microsoft/multicontainerwordpress
    ports:
      - "8000:80"
    restart: always

  redis:
    image: redis:3-alpine
    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_REDISHOST="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_REDISHOST",  
    "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|dmVyc2lvbjogJzMzMzMzY2Vz0gogICBteXNxbDoKICAgICBpbWFnZTogbXlzcWw6NS4CiAgICAgdm9sdW1lczoKICAgI  
CAgIC0gZGJfZGF0YTovdmFyL2xpYi9teXNxbAogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25tZW50gogICAgICAgTV1TUxfUk9  
PVF9QQVNTV09SRDogZXhhbXBsZXhc3MKCiAgIHdvcnRwcmVzczoKICAgICBkZXBlbmRzX29uOgogICAgICAgLSBteXNxbAogICAgIGltYWd1o  
iB3b3JkcHJlc3M6bGF0ZXN0CiAgICAgcG9ydHM6CiAgICAgICAtICI4MDAwOjgwIogICAgIHJlc3RhcnQ6IGFsd2F5cwogICAgIGVudmlyb25  
tZW50gogICAgICAgV09SRFBDRVNTX0RCX1BBU1NXT1JE0iBleGFtcGx1cGFzcwp2b2x1bWVz0gogICAgZGJfZGF0YTo="  
  }  
]
```

## 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**.

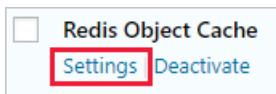
The screenshot shows the WordPress dashboard with the 'Plugins' menu item highlighted in red. The 'Installed Plugins' tab is selected. The dashboard features a 'Welcome to WordPress!' message and a 'Get Started' button. On the right, there's a sidebar with '1 Comment' and footer text indicating 'WordPress 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 with the 'All (3)' tab selected. The 'Redis Object Cache' plugin is listed, with its 'Activate' button highlighted in red. Other plugins listed include 'Akismet Anti-Spam' and 'Hello Dolly'. The page includes standard WordPress filtering and sorting tools like 'Bulk Actions' and 'Apply'.

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](#)

# Tutorial: Build a custom image and run in App Service from a private registry

1/28/2020 • 10 minutes to read • [Edit Online](#)

App Service provides built-in Docker images on Linux with support for specific versions, such as PHP 7.3 and Nodejs 10.14. App Service uses the Docker container technology to host both built-in images and custom images as a platform as a service. In this tutorial, you learn how to build a custom image and run it in App Service. This pattern is useful when the built-in images don't include your language of choice, or when your application requires a specific configuration that isn't provided within the built-in images.

In this tutorial, you learn how to:

- Deploy a custom image to a private container registry
- Run the custom image in App Service
- Configure environment variables
- Update and redeploy the image
- Access diagnostic logs
- Connect to the container using SSH

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

## Prerequisites

To complete this tutorial, you need:

- [Git](#)
- [Docker](#)

## Download the sample

In a terminal window, run the following command to clone the sample app repository to your local machine, then change to the directory that contains the sample code.

```
git clone https://github.com/Azure-Samples/docker-django-webapp-linux.git --config core.autocrlf=input  
cd docker-django-webapp-linux
```

## Build the image from the Docker file

In the Git repository, take a look at *Dockerfile*. This file describes the Python environment that is required to run your application. Additionally, the image sets up an [SSH](#) server for secure communication between the container and the host.

```
FROM python:3.4

RUN mkdir /code
WORKDIR /code
ADD requirements.txt /code/
RUN pip install -r requirements.txt
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"]
```

Build the Docker image with the `docker build` command.

```
docker build --tag mydockerimage .
```

Test that the build works by running the Docker container. Issue the `docker run` command and pass the name and tag of the image to it. Be sure to specify the port using the `-p` argument.

```
docker run -p 8000:8000 mydockerimage
```

Verify the web app and container are functioning correctly by browsing to <http://localhost:8000>.

The screenshot shows the Azure App Service landing page. At the top, there's a navigation bar with icons for back, forward, and refresh, followed by the URL 'localhost'. Below the header is a large banner with the text 'Build, deploy and scale applications faster' and a 'Learn more' button. In the background of the banner, there's a photo of two people working at a desk. Below the banner, the main heading is 'Get started with App Service on Linux'. There are four cards with icons and text: 'Overview' (document icon), 'Bring your own Docker Container' (cloud with Docker icon), 'Use Built-in images' (globe and penguin icon), and 'Samples and Resources' (cat icon). Each card has a small image of a person working on a computer.

## Get started with App Service on Linux

Overview    Bring your own Docker Container    Use Built-in images    Samples and Resources

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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

To create an app that uses the image you just created, you run Azure CLI commands that create a resource group, pushes the image, and then creates the App Service plan web app to run it.

## Create a resource group

A [resource group](#) is a logical container into which Azure resources like 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 Container Registry

In the Cloud Shell, use the `az acr create` command to create an Azure Container Registry.

```
az acr create --name <azure-container-registry-name> --resource-group myResourceGroup --sku Basic --admin-enabled true
```

## Sign in to Azure Container Registry

To push an image to the registry, you need to authenticate with the private registry. In the Cloud Shell, use the `az acr show` command to retrieve the credentials from the registry you created.

```
az acr credential show --name <azure-container-registry-name>
```

The output reveals two passwords along with the user name.

```
{
  "passwords": [
    {
      "name": "password",
      "value": "{password}"
    },
    {
      "name": "password2",
      "value": "{password}"
    }
  ],
  "username": "<registry-username>"
}
```

From your local terminal window, sign in to the Azure Container Registry using the `docker login` command, as shown in the following example. Replace *<azure-container-registry-name>* and *<registry-username>* with values for your registry. When prompted, type in one of the passwords from the previous step.

```
docker login <azure-container-registry-name>.azurecr.io --username <registry-username>
```

Confirm that the login succeeds.

## Push image to Azure Container Registry

Tag your local image for the Azure Container Registry. For example:

```
docker tag mydockerimage <azure-container-registry-name>.azurecr.io/mydockerimage:v1.0.0
```

Push the image by using the `docker push` command. Tag the image with the name of the registry, followed by your image name and tag.

```
docker push <azure-container-registry-name>.azurecr.io/mydockerimage:v1.0.0
```

Back in the Cloud Shell, verify that the push is successful.

```
az acr repository list -n <azure-container-registry-name>
```

You should get the following output.

```
[  
  "mydockerimage"  
]
```

## Create 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 **Free** pricing tier (`--sku F1`) and in a Linux container (`--is-linux`).

```
az appservice plan create --name myAppServicePlan --resource-group myResourceGroup --sku F1 --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": "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 web app

In the Cloud Shell, create a **web app** in the `myAppServicePlan` App Service plan with the `az webapp create` command. Replace `<app-name>` with a unique app name, and `<azure-container-registry-name>` with your registry name.

```
az webapp create --resource-group myResourceGroup --plan myAppServicePlan --name <app-name> --deployment-container-image-name <azure-container-registry-name>.azurecr.io/mydockerimage:v1.0.0
```

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

```
{  
    "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. >  
}
```

## Configure registry credentials in web app

For App Service to pull the private image, it needs information about your registry and image. In the Cloud Shell, provide them with the `az webapp config container set` command. Replace `<app-name>`, `<azure-container-registry-name>`, `<registry-username>`, and `<password>`.

```
az webapp config container set --name <app-name> --resource-group myResourceGroup --docker-custom-image-name <azure-container-registry-name>.azurecr.io/mydockerimage:v1.0.0 --docker-registry-server-url https://<azure-container-registry-name>.azurecr.io --docker-registry-server-user <registry-username> --docker-registry-server-password <password>
```

### NOTE

When using a registry other than Docker Hub, `--docker-registry-server-url` must be formatted as `https://` followed by the fully qualified domain name of the registry.

## Configure environment variables

Most Docker images use custom environment variables, such as a port other than 80. You tell App Service about the port that your image uses by using the `WEBSITES_PORT` app setting. The GitHub page for the [Python sample in this tutorial](#) shows that you need to set `WEBSITES_PORT` to `8000`.

To set app settings, use the `az webapp config appsettings set` command in the Cloud Shell. App settings are case-sensitive and space-separated.

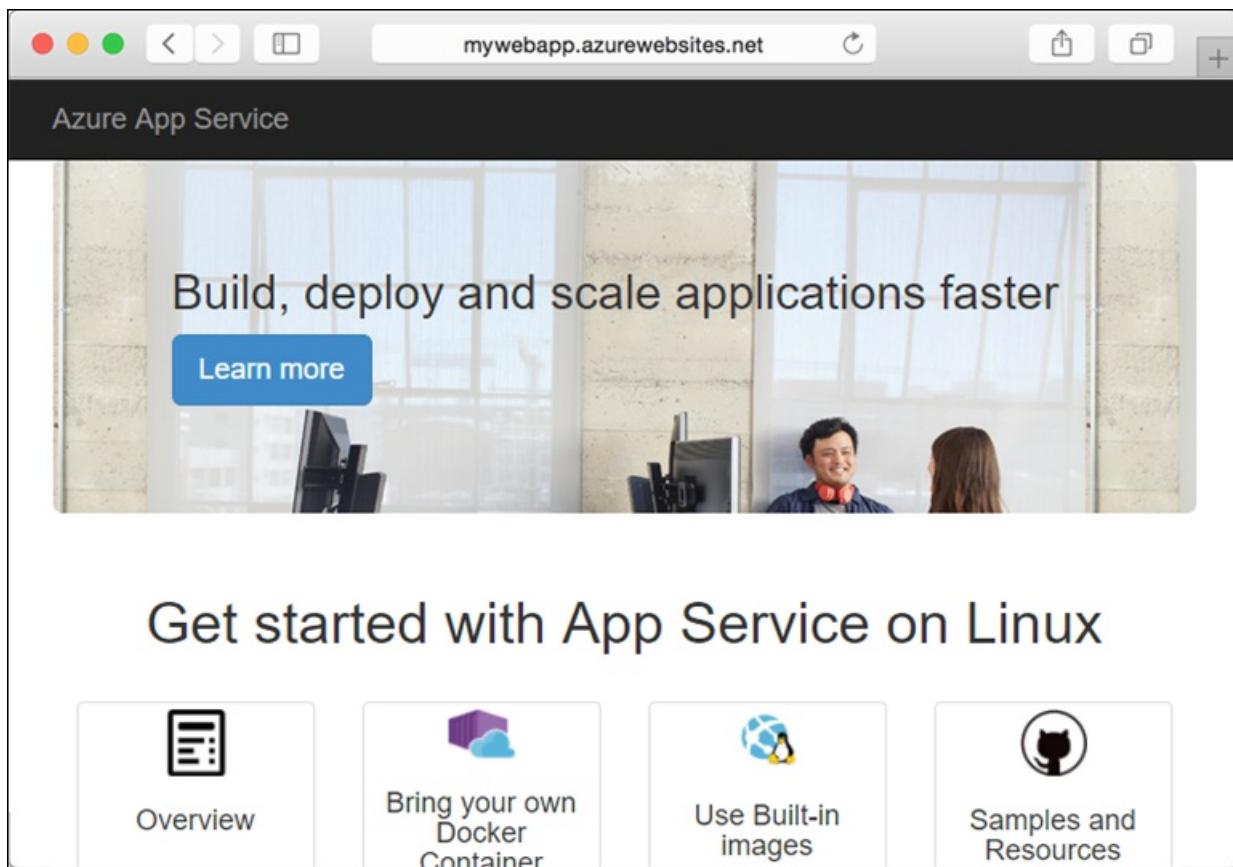
```
az webapp config appsettings set --resource-group myResourceGroup --name <app-name> --settings  
WEBSITES_PORT=8000
```

## Test the web app

Verify that the web app works by browsing to it (`http://<app-name>.azurewebsites.net`).

### NOTE

The first time you access the app, it may take some time because App Service needs to pull the entire image. If the browser times out, just refresh the page.



The screenshot shows a Mac OS X browser window with the URL "mywebapp.azurewebsites.net" in the address bar. The page title is "Azure App Service". The main content features a large banner with the text "Build, deploy and scale applications faster" and a "Learn more" button. Below the banner, there's a photo of two people working at desks in an office. The section "Get started with App Service on Linux" follows, with four cards: "Overview" (document icon), "Bring your own Docker Container" (cloud icon), "Use Built-in images" (globe and penguin icon), and "Samples and Resources" (GitHub logo). Each card has a small "Get Started" button.

## Change web app and redeploy

In your local Git repository, open app/templates/app/index.html. Locate the first HTML element and change it to.

```
<nav class="navbar navbar-inverse navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Azure App Service - Updated Here!</a>
    </div>
  </div>
</nav>
```

Once you've modified the Python file and saved it, you must rebuild and push the new Docker image. Then restart the web app for the changes to take effect. Use the same commands that you have previously used in this tutorial. You can refer to [Build the image from the Docker file](#) and [Push image to Azure Container Registry](#). Test the web app by following the instructions in [Test the web app](#).

## Access diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

**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`.

## Enable SSH connections

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. Let's take a look at the sample repository that already has the necessary configuration.

- In the [Dockerfile](#), the following code installs the SSH server and also sets the sign-in credentials.

```
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 does not 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.

- The [Dockerfile](#) copies the [sshd\\_config](#) file in the repository to the `/etc/ssh/` directory.

```
COPY sshd_config /etc/ssh/
```

- The [Dockerfile](#) exposes port 2222 in the container. It is an internal port accessible only by containers within the bridge network of a private virtual network.

```
EXPOSE 8000 2222
```

- The [entry script](#) starts the SSH server.

```
#!/bin/bash  
service ssh start
```

### Open SSH connection to container

SSH connection is available only through the Kudu site, which is accessible at

<https://<app-name>.scm.azurewebsites.net>.

To connect, browse to <https://<app-name>.scm.azurewebsites.net/webssh/host> and sign in with your Azure account.

You are then redirected to a page displaying an interactive console.

You may wish to verify that certain applications are running in the container. To inspect the container and verify running processes, issue the `top` command at the prompt.

```
top
```

The `top` command exposes all running processes in a container.

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	945616	35372	15348	S	0.0	2.1	0:04.63	node
20	root	20	0	55180	2776	2516	S	0.0	0.2	0:00.00	sshd
42	root	20	0	944596	33340	15352	S	0.0	1.9	0:05.80	node /opt/s+
56	root	20	0	59812	5244	4512	S	0.0	0.3	0:00.93	sshd
58	root	20	0	20228	3128	2664	S	0.0	0.2	0:00.00	bash
62	root	20	0	21916	2272	1944	S	0.0	0.1	0:03.15	top
63	root	20	0	59812	5344	4612	S	0.0	0.3	0:00.03	sshd
65	root	20	0	20228	3140	2672	S	0.0	0.2	0:00.00	bash
71	root	20	0	59812	5380	4648	S	0.0	0.3	0:00.02	sshd
73	root	20	0	20228	3160	2696	S	0.0	0.2	0:00.00	bash
77	root	20	0	21920	2304	1972	R	0.0	0.1	0:00.00	top

Congratulations! You've configured a custom Linux container in App Service.

## 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

What you learned:

- Deploy a custom image to a private container registry
- Run the custom image in App Service
- Configure environment variables
- Update and redeploy the image
- Access diagnostic logs
- Connect to the container using SSH

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:

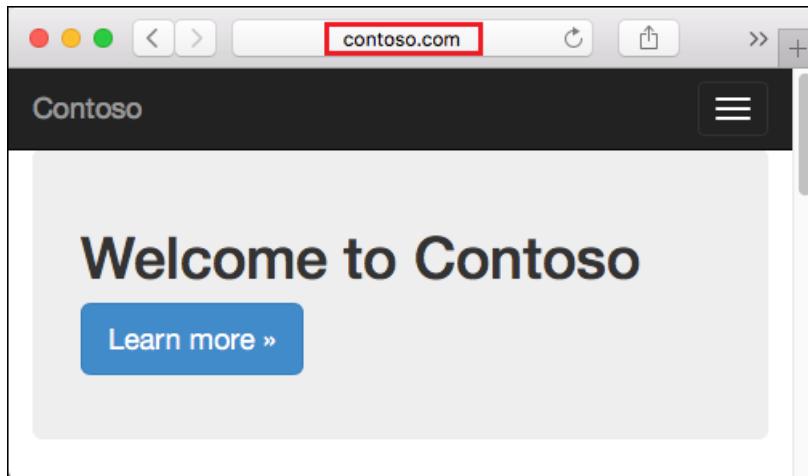
[Configure custom container](#)

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

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

12/2/2019 • 12 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 DNS name to Azure App Service.



In this tutorial, you learn how to:

- Map a subdomain (for example, `www.contoso.com`) by using a CNAME record
- Map a root domain (for example, `contoso.com`) by using an A record
- Map a wildcard domain (for example, `*.contoso.com`) by using a CNAME record
- Redirect the default URL to a custom directory
- Automate domain mapping with scripts

## Prerequisites

To complete this tutorial:

- [Create an App Service app](#), or use an app that you created for another tutorial.
- Purchase a domain name and make sure you have 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.

### NOTE

If you don't have an existing domain name, consider [purchasing a domain using the Azure portal](#).

## 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

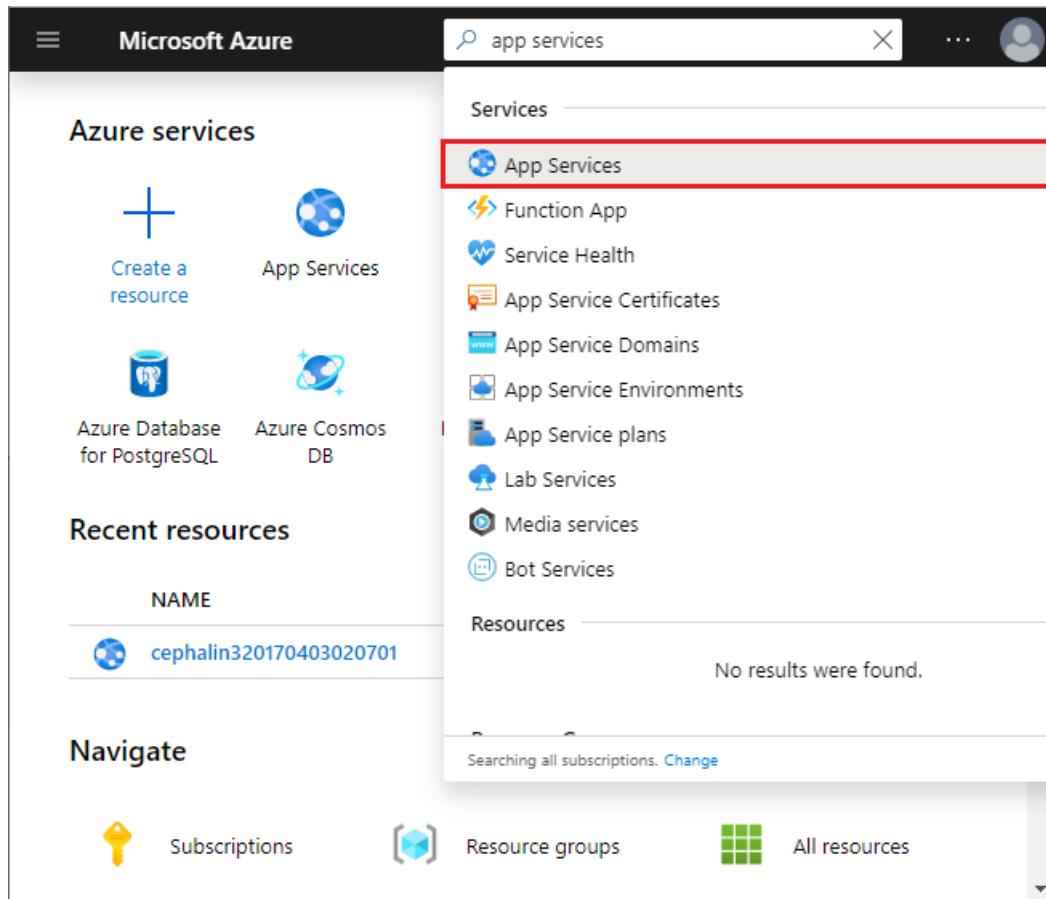
App Service Free and Shared (preview) hosting plans are base tiers that run on the same Azure virtual machines as other App Service apps. Some apps might belong to other customers. These tiers are intended to be used only for development and testing purposes.

## Sign in to Azure

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

## Select the app in the Azure portal

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, a sidebar on the left lists "Azure services" including "Create a resource", "App Services", "Azure Database for PostgreSQL", and "Azure Cosmos DB". Under "Recent resources", there's a table with one item: "cephalin320170403020701". On the right, the main content area has a heading "Services" and a list of items. 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 this list, under "Resources", it says "No results were found". At the bottom of the page, there's a "Navigate" section with links for "Subscriptions", "Resource groups", and "All resources".

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

Home > App Services

## App Services

Microsoft

Documentation

**Subscriptions:** All 2 selected – Don't see a subscription? [Open Directory + Subscription settings](#)

Filter by name... All subsc... All resou... All locati... All tags No group...

6 items

<input type="checkbox"/> Name ↑↓	Status	App Type	App Service
<input type="checkbox"/> cephalin320170403020701	Running	Web App	test-sku
<input type="checkbox"/> denniseastusbot	Running	Web App	z76-z763
<input type="checkbox"/> myFirstAzureWebApp20190...	Running	Web App	ServicePl
<input type="checkbox"/> WebApplicationASPDotNET...	Running	Web App	ServicePl

You see the management page of the App Service app.

### Check the pricing tier

In the left navigation of the app page, scroll to the **Settings** section and 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 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

If the App Service plan is not in the **F1** tier, close the **Scale up** page and skip to [Map a CNAME record](#).

### Scale up the App Service plan

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**.

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 your domain

You can use either a **CNAME record** or an **A record** to map a custom DNS name to App Service. Follow the respective steps:

- [Map a CNAME record](#)
- [Map an A record](#)

- Map a wildcard domain (with a CNAME record)

#### NOTE

You should use CNAME records for all custom DNS names except root domains (for example, `contoso.com`). For root domains, use A records.

## Map a CNAME record

In the tutorial example, you add a CNAME record for the `www` subdomain (for example, `www.contoso.com`).

### Access DNS records with domain provider

#### NOTE

You can use Azure DNS to configure a custom DNS name for your Azure Web Apps. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

Sign in to the website of your domain provider.

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 is 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, you 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.

### Create the CNAME record

Add a CNAME record to map a subdomain to the app's default domain name (`<app_name>.azurewebsites.net`, where `<app_name>` is the name of your app).

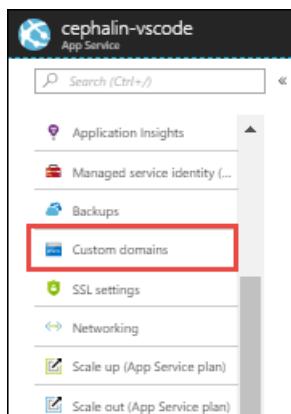
For the `www.contoso.com` domain example, add a CNAME record that maps the name `www` to `<app_name>.azurewebsites.net`.

After you add the CNAME, the DNS records page looks like the following example:

Records				
Last updated 6/18/2018 3:43 PM				
Type	Name	Value	TTL	
CNAME	www	cephalin-vscode.azurewebsites.net	1 Hour	
				<b>ADD</b>

#### Enable the CNAME record mapping in Azure

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



In the **Custom domains** page of the app, add the fully qualified custom DNS name (`www.contoso.com`) to the list.

Select the + icon next to **Add custom domain**.

SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
	my-demo-app.azurewebsites.net	

Type the fully qualified domain name that you added a CNAME record for, such as `www.contoso.com`.

Select **Validate**.

The **Add custom domain** page is shown.

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. [Learn More](#)

CNAME  
my-demo-app.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. Try refreshing the browser to update the data.

**Custom Domains**

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

IP address: 13.69.68.6

HTTPS Only: Off

**Add custom domain**

Status Filter: All (2) Not Secure (1) Secure (1)

SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
Not Secure	www.contoso.com	Add binding
Secure	my-demo-app.azurewebsites.net	

#### NOTE

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

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

**Hostname availability**

**Domain ownership**

Create an CNAME record with your DNS provider using the configuration below. [Learn more](#)

Type	Host	Value
CNAME	www or subdomain	cephalin-vscode.azurewebsites.net

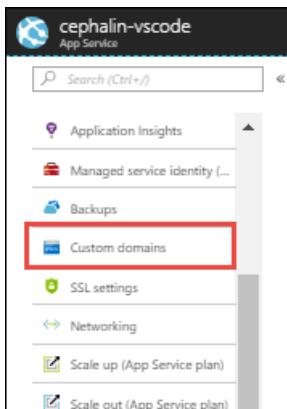
#### Map an A record

In the tutorial example, you add an A record for the root domain (for example, `contoso.com`).

#### Copy the app's IP address

To map an A record, you need the app's external IP address. You can find this IP address in the app's **Custom domains** page in the Azure portal.

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



In the **Custom domains** page, copy the app's IP address.

A screenshot of the 'Custom Hostnames' configuration page. It shows an IP address of 40.118.102.46 and an 'HTTPS Only' toggle switch set to 'Off'. Below this, there's a section for 'HOSTNAMES ASSIGNED TO SITE' which lists 'cephalin-vscode.azurewebsites.net'. There's also a 'SSL BINDING' section.

#### Access DNS records with domain provider

##### NOTE

You can use Azure DNS to configure a custom DNS name for your Azure Web Apps. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

Sign in to the website of your domain provider.

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 is named something like **Zone file**, **DNS Records**, or **Advanced configuration**.

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

A screenshot of a DNS records management interface. At the top, it says 'Records' and 'Last updated 6/18/2018 3:40 PM'. Below is a table with columns: Type, Name, Value, and TTL. The table contains three rows: an NS record for '@' pointing to ns31.domaincontrol.com with a 1 Hour TTL, another NS record for '@' pointing to ns32.domaincontrol.com with a 1 Hour TTL, and an SOA record for '@' with a Primary nameserver value of ns31.domaincontrol.com and a 600 seconds TTL. At the bottom right is a blue 'ADD' button.

In the example screenshot, you 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.

#### Create the A record

To map an A record to an app, App Service requires **two** DNS records:

- An **A** record to map to the app's IP address.
- A **TXT** record to map to the app's default domain name `<app_name>.azurewebsites.net`. App Service uses this record only at configuration time, to verify that you own the custom domain. After your custom domain is validated and configured in App Service, you can delete this TXT record.

For the `contoso.com` domain example, create the A and TXT records according to the following table (@ typically represents the root domain).

RECORD TYPE	HOST	VALUE
A	@	IP address from <a href="#">Copy the app's IP address</a>
TXT	@	<code>&lt;app_name&gt;.azurewebsites.net</code>

#### NOTE

To add a subdomain (like `www.contoso.com`) using an A record instead of a recommended **CNAME record**, your A record and TXT record should look like the following table instead:

RECORD TYPE	HOST	VALUE
A	www	IP address from <a href="#">Copy the app's IP address</a>
TXT	www	<code>&lt;app_name&gt;.azurewebsites.net</code>

When the records are added, the DNS records page looks like the following example:

### Records

Last updated 6/18/2018 3:50 PM

Type	Name	Value	TTL	
A	@	40.118.102.46	1 Hour	
TXT	@	cephalin-vscode.azurewebsites.net	1 Hour	

**ADD**

#### Enable the A record mapping in the app

Back in the app's **Custom domains** page in the Azure portal, add the fully qualified custom DNS name (for example, `contoso.com`) to the list.

Select the + icon next to **Add custom domain**.

The screenshot shows the 'Custom Domains' section of the Azure portal. At the top, there's a globe icon and the title 'Custom Domains'. Below it, a sub-header says 'Configure and manage custom domains assigned to your app' with a 'Learn more' link. There are two input fields: 'IP address:' containing '13.69.68.6' and 'HTTPS Only:' with a toggle switch set to 'Off'. A large blue '+' button labeled 'Add custom domain' is prominently displayed. Below these are 'Status Filter' buttons for 'All (1)', 'Not Secure (0)', and 'Secure (1)'. A table header row includes 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. Under 'SSL STATE', there's a green checkmark for 'Secure' and a link to 'my-demo-app.azurewebsites.net'. The table has one data row.

Type the fully qualified domain name that you configured the A record for, such as `contoso.com`.

Select **Validate**.

The **Add custom domain** page is shown.

Make sure that **Hostname record type** is set to **A record (example.com)**.

Select **Add custom domain**.

This is a modal dialog titled 'Add custom domain' for 'my-demo-app'. It contains several input fields and buttons. The 'Custom domain' field is filled with 'contoso.com' and has a red border. Below it is a 'Validate' button with a red border. A dropdown menu for 'Hostname record type' is open, showing 'A record (example.com)' with a red border. At the bottom, there's an 'A record configuration' section with a globe icon, a note about A records mapping domains to IP addresses, an 'External IP address' field containing '13.69.68.6', and a final 'Add custom domain' button with a red border.

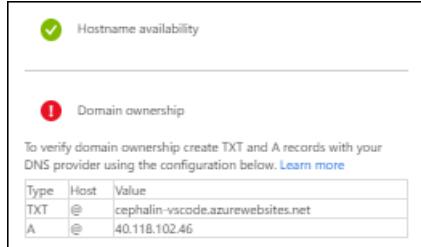
It might take some time for the new custom domain to be reflected in the app's **Custom domains** page. Try refreshing the browser to update the data.

The screenshot shows the 'Custom Domains' page again. The 'SSL STATE' filter is now set to 'Not Secure (1)', which highlights the 'contoso.com' entry. This entry shows a red exclamation mark icon, indicating that while the domain is listed, it is not yet secure. The 'Secure' entry for 'my-demo-app.azurewebsites.net' remains green with a checkmark. Other interface elements like the IP address input, HTTPS toggle, and status filters are visible at the top.

## NOTE

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

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



## Map a wildcard domain

In the tutorial example, you map a **wildcard DNS name** (for example, `*.contoso.com`) to the App Service app by adding a CNAME record.

### Access DNS records with domain provider

## NOTE

You can use Azure DNS to configure a custom DNS name for your Azure Web Apps. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

Sign in to the website of your domain provider.

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 is named something like **Zone file, DNS Records, or Advanced configuration**.

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

Type	Name	Value	TTL
NS	@	ns31.domaincontrol.com	1 Hour
NS	@	ns32.domaincontrol.com	1 Hour
SOA	@	Primary nameserver: ns31.domaincontrol.com.	600 seconds
<b>ADD</b>			

In the example screenshot, you 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.

### Create the CNAME record

Add a CNAME record to map a wildcard name to the app's default domain name (`<app_name>.azurewebsites.net`).

For the `*.contoso.com` domain example, the CNAME record will map the name `*` to `<app_name>.azurewebsites.net`.

When the CNAME is added, the DNS records page looks like the following example:

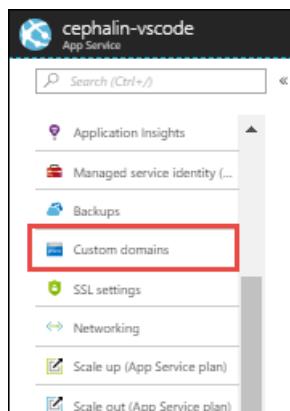
Type	Name	Value	TTL
CNAME	*	cephalin-vscode.azurewebsites.net	1 Hour

**ADD**

### Enable the CNAME record mapping in the app

You can now add any subdomain that matches the wildcard name to the app (for example, `sub1.contoso.com` and `sub2.contoso.com` match `*.contoso.com`).

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



Select the + icon next to **Add custom domain**.

SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
Secure	my-demo-app.azurewebsites.net	
Not Secure	my-test-domain.com	

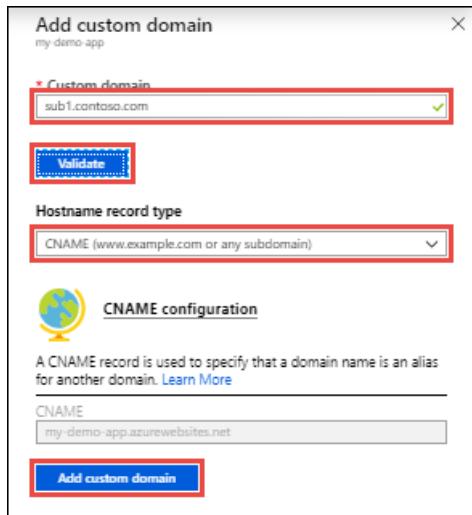
Type a fully qualified domain name that matches the wildcard domain (for example, `sub1.contoso.com`), and then

select **Validate**.

The **Add custom domain** button is activated.

Make sure that **Hostname record type** is set to **CNAME record (www.example.com or any subdomain)**.

Select **Add custom domain**.



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

Select the + icon again to add another custom domain that matches the wildcard domain. For example, add `sub2.contoso.com`.

The screenshot shows the 'Custom Domains' management page for a specific app. It includes the following sections:

- Custom Domains:** A title with a globe icon.
- Configure and manage custom domains assigned to your app.** A link to 'Learn more'.
- IP address:** Set to '13.69.68.6'.
- HTTPS Only:** A toggle switch set to 'Off'.
- Add custom domain:** A button with a '+' icon.
- Status Filter:** Buttons for 'All (3)', 'Not Secure (2)', and 'Secure (1)'.
- SSL STATE:** A table showing custom domains and their SSL status:

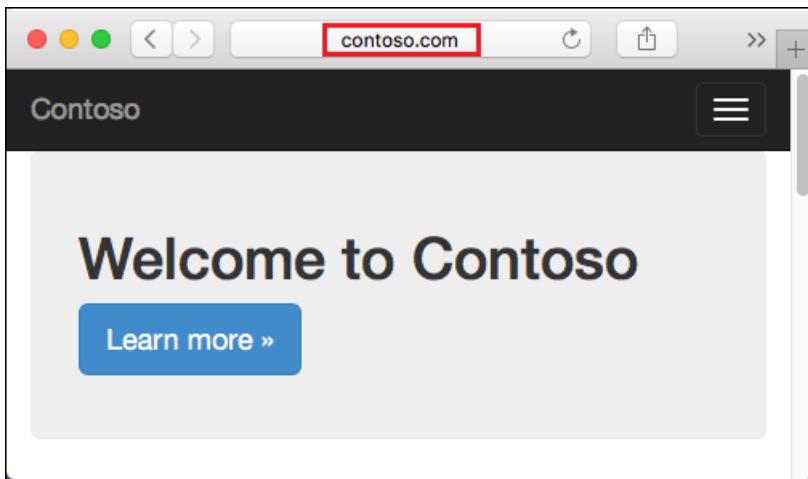
SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
! Not Secure	sub1.contoso.com	Add binding
! Not Secure	sub2.contoso.com	Add binding
✓ Secure	my-demo-app.azurewebsites.net	

#### NOTE

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

## Test in browser

Browse to the DNS name(s) that you configured earlier (for example, `contoso.com`, `www.contoso.com`, `sub1.contoso.com`, and `sub2.contoso.com`).



## Resolve 404 “Not Found”

If you receive an HTTP 404 (Not Found) error when browsing to the URL of your custom domain, verify that your domain resolves to your app's IP address using [WhatsmyDNS.net](#). If not, it may be due to one of the following reasons:

- The custom domain configured is missing an A record and/or a CNAME record.
- 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

By default, App Service directs web requests to the root directory of your app code. However, 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 would be accessible at `http://contoso.com/public`, but you would really want to direct `http://contoso.com` to the `public` directory instead. This step doesn't involve DNS resolution, but customizing the virtual directory.

To do this, select **Application settings** in the left-hand navigation of your web app page.

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.

The screenshot shows the 'Application settings' page for an Azure App Service. The left sidebar lists various settings: Application settings (selected and highlighted with a red box), Authentication / Authorization, Managed service identity, Backups, Custom domains, SSL certificates, Networking, and Scale up (App Service plan). The main area contains sections for 'hostingstart.html', 'Handler mappings' (with a note 'No results'), and 'Virtual applications and directories'. In the 'Virtual applications and directories' section, there is a row with '/ site\wwwroot\public' checked under 'Application'. A red box highlights the 'site\wwwroot\public' path.

Once the operation completes, your app should return the right page at the root path (for example, <http://contoso.com> ).

## Automate with scripts

You can automate management of custom domains with scripts, 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 new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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

In this tutorial, you learned 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
- Automate domain mapping with scripts

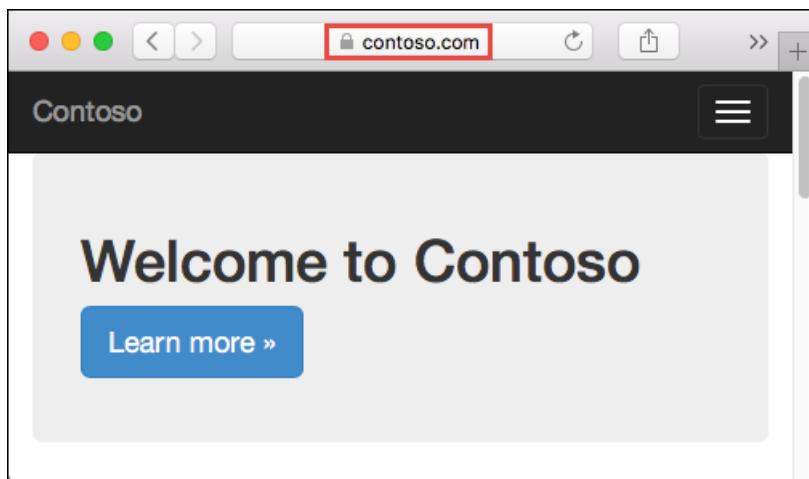
Advance to the next tutorial to learn how to bind a custom SSL certificate to a web app.

[Secure a custom DNS name with an SSL binding in Azure App Service](#)

# Secure a custom DNS name with an SSL binding in Azure App Service

12/4/2019 • 7 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 [requirements for SSL bindings](#).
- Create an SSL 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 \(Preview\)](#).

## Prepare your web app

To bind a custom SSL certificate (a third-party certificate or App Service certificate) to your web app, your [App](#)

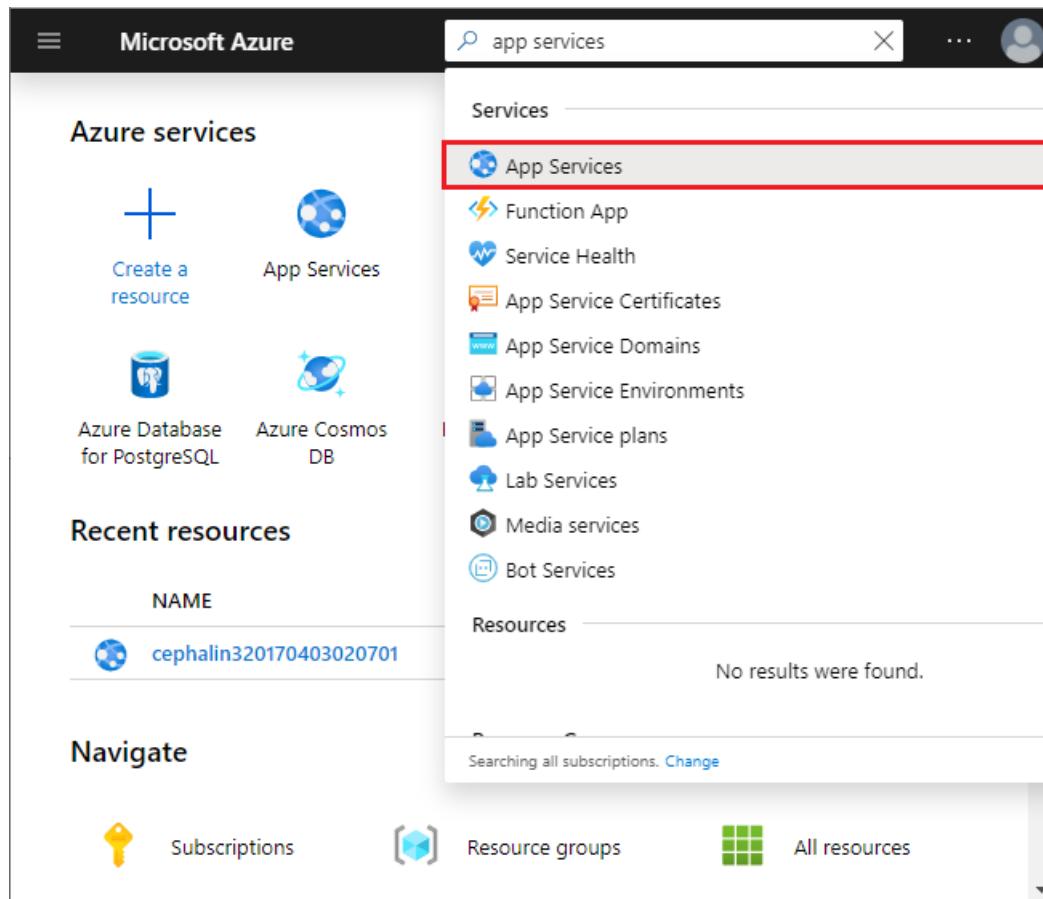
Service plan must be in the **Basic, Standard, Premium**, or **Isolated** tier. In this step, you make sure that your web app is in the supported pricing tier.

## Sign in to Azure

Open the [Azure portal](#).

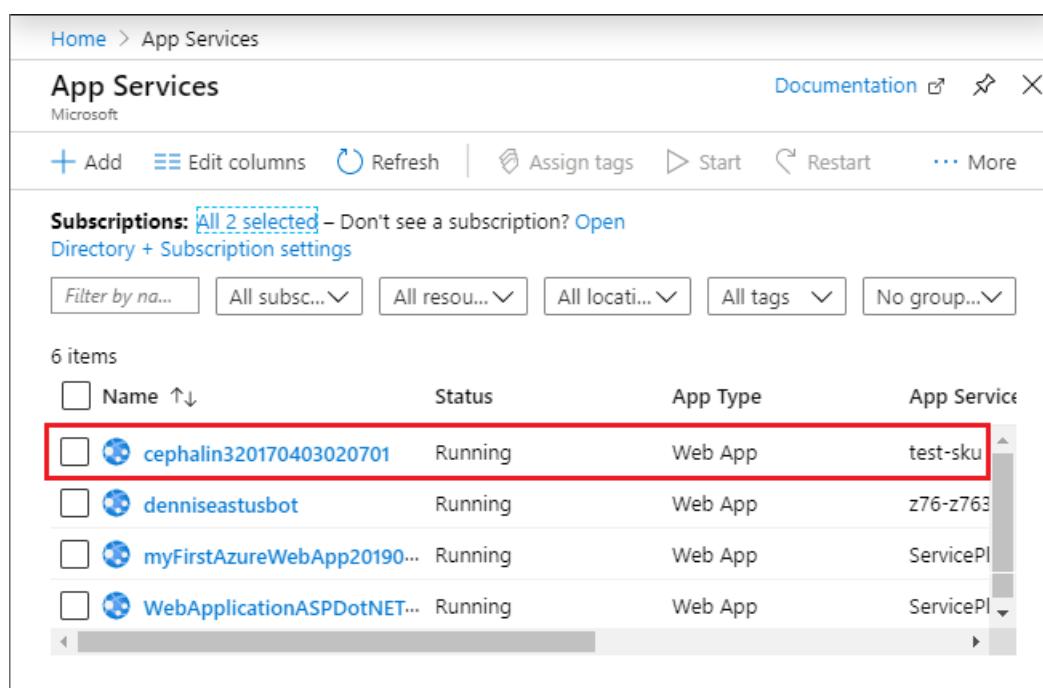
## Navigate to your web app

Search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. The search bar at the top contains the text "app services". Below the search bar, the "Services" section is expanded, showing a list of options: App Services (highlighted with a red box), 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 links for "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 listed: "cephalin320170403020701". At the bottom of the page 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 in the Azure portal. The page title is "App Services" under the "Microsoft" category. At the top, there are buttons for "Add", "Edit columns", "Refresh", "Assign tags", "Start", "Restart", and "More". Below this is a "Subscriptions" section with a note about seeing all subscriptions. There are also filter buttons for "Filter by na...", "All subsc...", "All resou...", "All locati...", "All tags", and "No group...". The main table displays 6 items, each with a checkbox, name, status, app type, and app service. One row, "cephalin320170403020701", is highlighted with a red box. The table columns are: Name, Status, App Type, and App Service. The "Name" column includes a sorting arrow icon.

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-hand navigation pane of the Azure App Service settings. At the top, there's a profile icon and the name 'cephalin320170403020701' followed by 'App Service'. Below this is a search bar labeled 'Search (Ctrl+ /)'. The main navigation menu is titled 'Settings' and includes the following items:

- Deployment Center
- 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 rectangular 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**

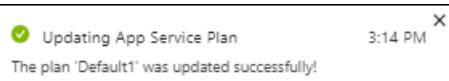
Memory per instance available to run applications deployed and running in...

**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.



### 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**

The screenshot shows the 'Custom Domains' section of the Azure portal. It displays two assigned custom domains: 'contoso.com' (marked as 'Not Secure') and 'my-demo-app.azurewebsites.net' (marked as 'Secure'). Below the domain list is a table with columns for 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. The 'SSL BINDING' column contains a 'Add binding' button, which is highlighted with a red box. Other buttons in this column include 'Edit' and 'Delete'.

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](#) (Preview) 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 SSL binding in the **TLS/SSL Binding** dialog, then click **Add Binding**.

SETTING	DESCRIPTION
Custom domain	The domain name to add the SSL binding for.
Private Certificate Thumbprint	The certificate to bind.

SETTING	DESCRIPTION
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 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 SSL certificate to secure a dedicated public IP address. After you configure the binding, follow the steps in <a href="#">Remap A record for IP SSL</a>. IP SSL is supported only in Production or Isolated tiers.</li> </ul>

Once the operation is complete, the custom domain's SSL state is changed to **Secure**.

Status Filter <span>All (2)</span> Not Secure (0) Secure (2)			
SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING	
<span>Secure</span>	contoso.com	SNI SSL	...
<span>Secure</span>	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 A record for IP SSL

If you don't use IP SSL in your app, skip to [Test HTTPS for your custom domain](#).

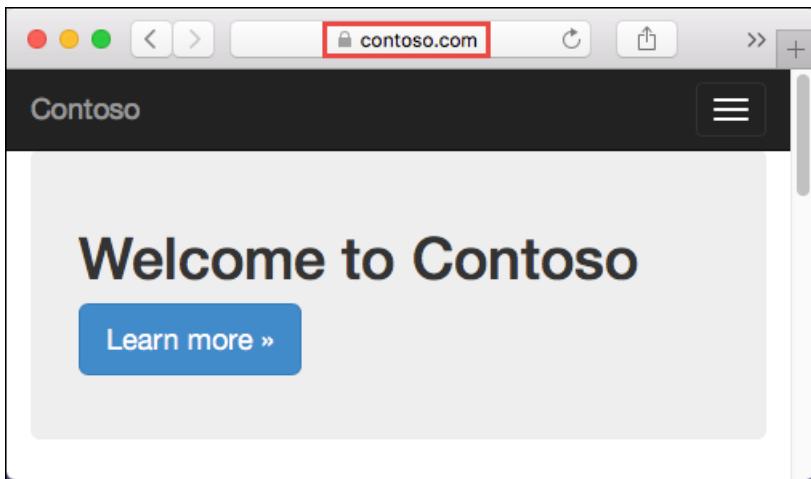
By default, your app uses a shared public IP address. When you bind a certificate with IP SSL, App Service creates a new, dedicated IP address for your app.

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.

## 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

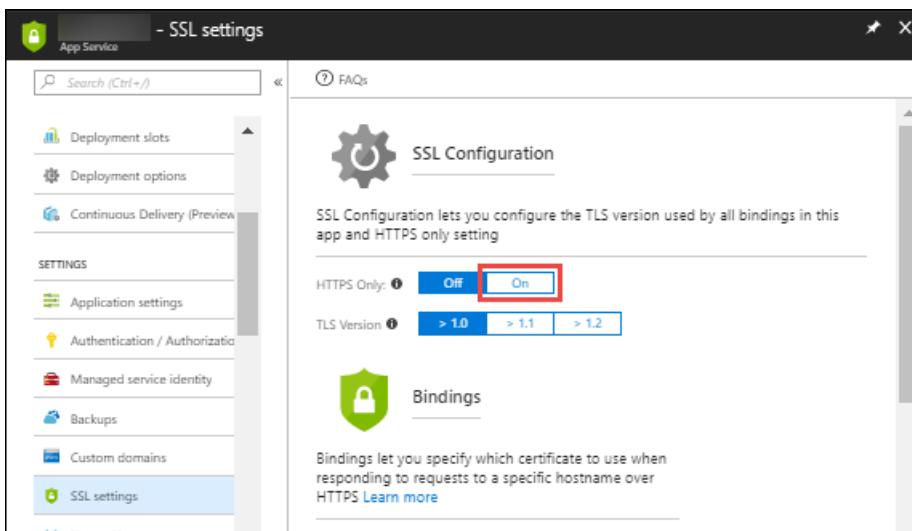
Your inbound IP address can change when you delete a binding, even if that binding is IP SSL. This is especially important when you renew a certificate that's already in an IP SSL binding. To avoid a change in your app's IP address, follow these steps in order:

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 **SSL settings**. Then, in **HTTPS Only**, select **On**.



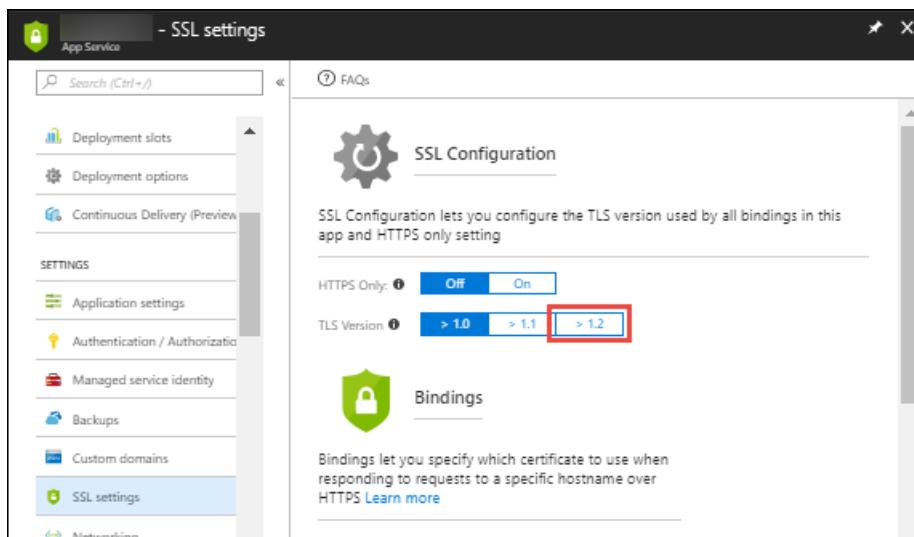
When the operation is complete, navigate to any of the HTTP URLs that point to your app. For example:

- `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 **SSL settings**. Then, in **TLS version**, select the minimum TLS version you want. This setting controls the inbound calls only.



When the operation is complete, your app rejects all connections with lower TLS versions.

## 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

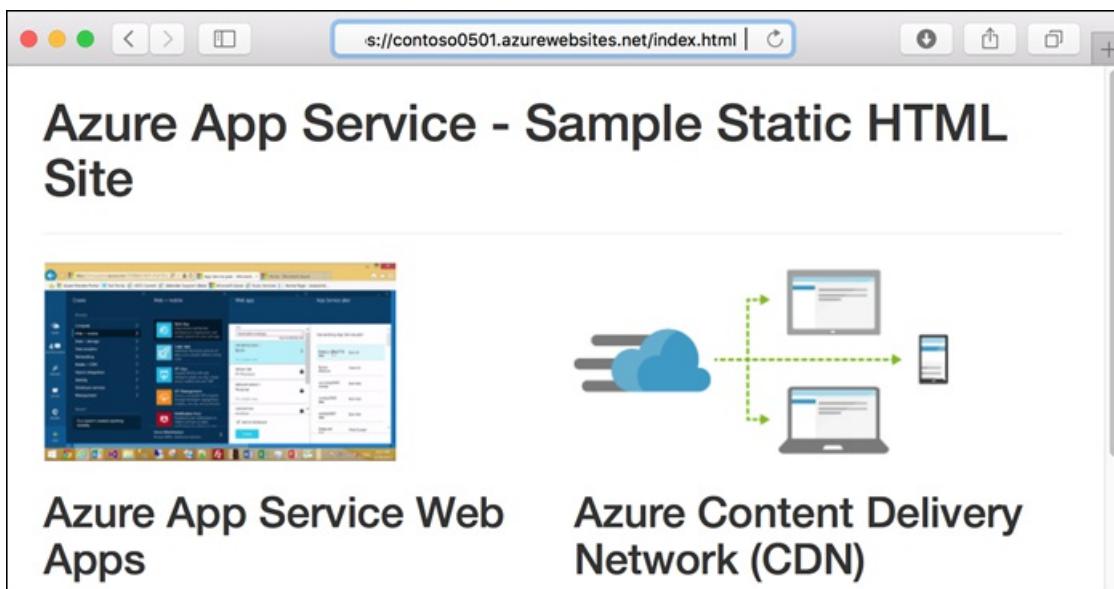
- [Use an SSL certificate in your application code](#)
- [FAQ : App Service Certificates](#)

# Tutorial: Add Azure CDN to an Azure App Service web app

7/5/2019 • 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](#).

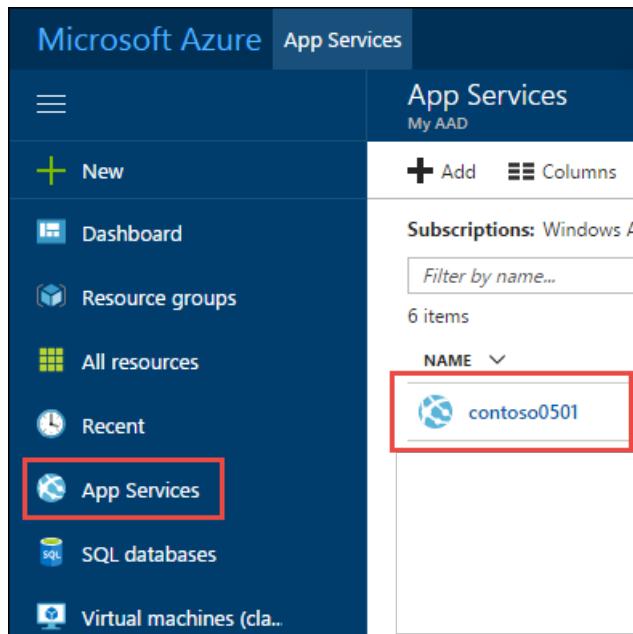
### Dynamic site acceleration optimization

If you want to optimize your CDN endpoint for dynamic site acceleration (DSA), you should use the [CDN portal](#) to

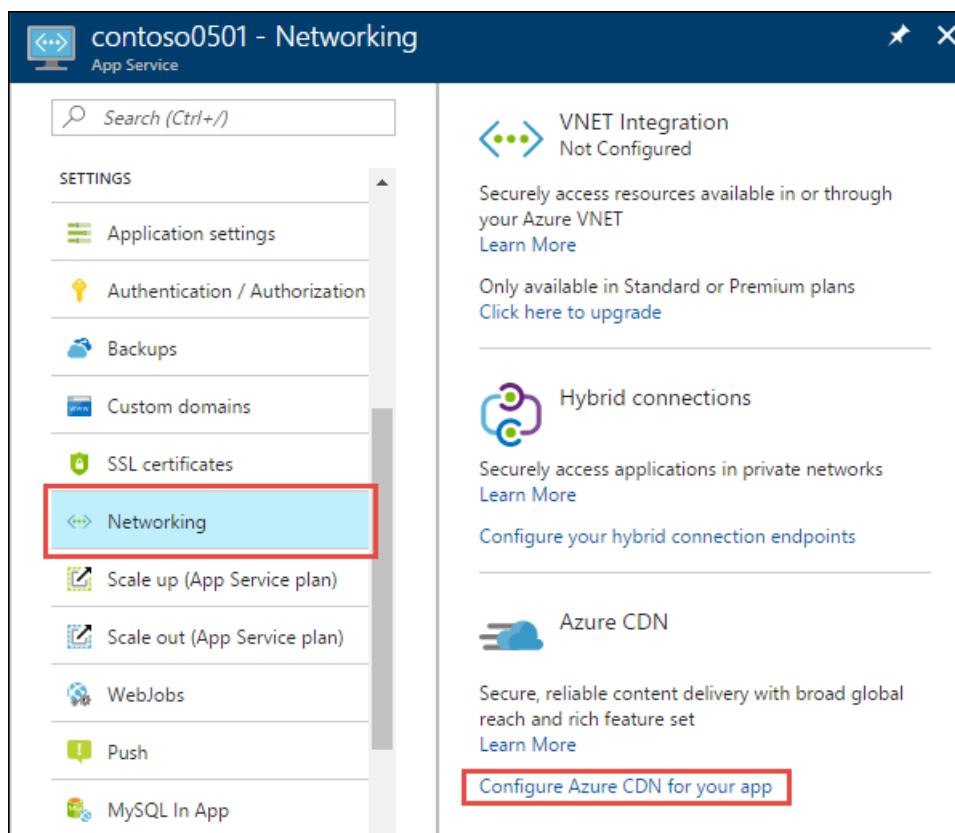
create your profile and endpoint. With [DSA optimization](#), the performance of web pages with dynamic content is measurably improved. For instructions about how to optimize a CDN endpoint for DSA from the CDN portal, see [CDN endpoint configuration to accelerate delivery of dynamic files](#). Otherwise, 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](#).



In the **App Service** page, in the **Settings** section, select **Networking > Configure Azure CDN for your app**.



In the **Azure Content Delivery Network** page, provide the **New endpoint** settings as specified in the table.

Azure CDN

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
There are no CDN endpoints linked to your resource. You may create new endpoints below.		

### New endpoint

CDN profile [?](#)

Create new  Use existing

myCDNProfile ✓

\* Pricing tier [\(View full pricing details\)](#)

Standard Akamai ▼

\* CDN endpoint name [?](#)

contoso0501 ✓

.azureedge.net

Origin hostname [?](#)

contoso0501.azurewebsites.net

**Create**

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>CDN profile</b>	myCDNProfile	A CDN profile is a collection of CDN endpoints with the same pricing tier.
<b>Pricing tier</b>	Standard Akamai	The <a href="#">pricing tier</a> specifies the provider and available features. This tutorial uses <i>Standard Akamai</i> .
<b>CDN endpoint name</b>	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.html` 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 master
```

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`).

Microsoft Azure Resource groups

New Dashboard Resource groups All resources Recent

Resource groups

Subscriptions: Windows Azure MSDN - Visual Studio Ultimate

Filter by name...

7 items

NAME

myResourceGroup

In the list of resources, select your CDN endpoint.

myResourceGroup Resource group

Search (Ctrl+ /)

Overview Activity log Access control (IAM) Tags

SETTINGS

Quickstart Resource costs Deployments Properties

Add Columns Delete Refresh Move

Subscription name (change) Windows Azure MSDN - Visual Studio Ultim.. Subscription ID

Deployments 1 Succeeded

Filter by name...

4 items

NAME	TYPE
contoso0501	App Service
contosocdnprofile	CDN profile
contoso0501	Endpoint
quickStartPlan	App Service plan

At the top of the **Endpoint** page, select **Purge**.

contoso0501 Endpoint

Search (Ctrl+ /)

Overview Activity log Access control (IAM)

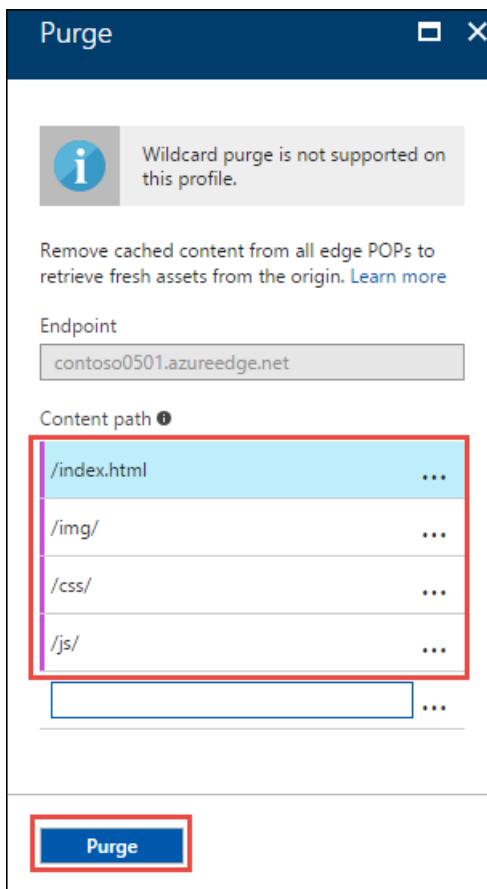
Custom domain Purge Load Stop

Essentials

Resource group myResourceGroup Status Running

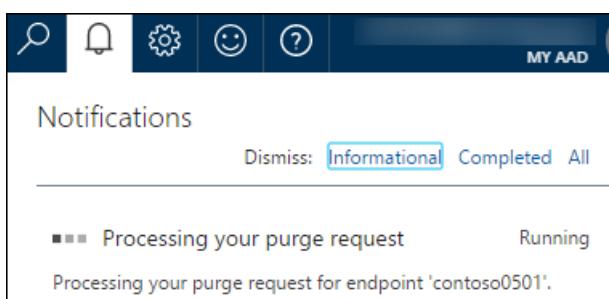
Enter the content paths you want to purge. You can pass a complete file path to purge an individual file, or a path segment to purge and refresh all content in a folder. Because you changed *index.html*, ensure that is in one of the paths.

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.

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**

**Overview**

**Activity log**

**Access control (IAM)**

**Tags**

**Diagnose and solve problems**

**SETTINGS**

**Origin**

**Custom domains**

**Compression**

**Cache**

**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 master
```

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:

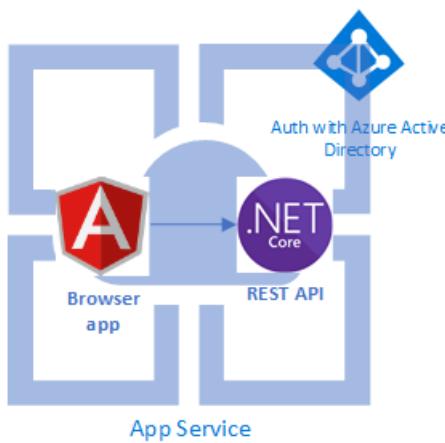
[Tutorial: Add a custom domain to your Azure CDN endpoint](#)

# Tutorial: Authenticate and authorize users end-to-end in Azure App Service on Linux

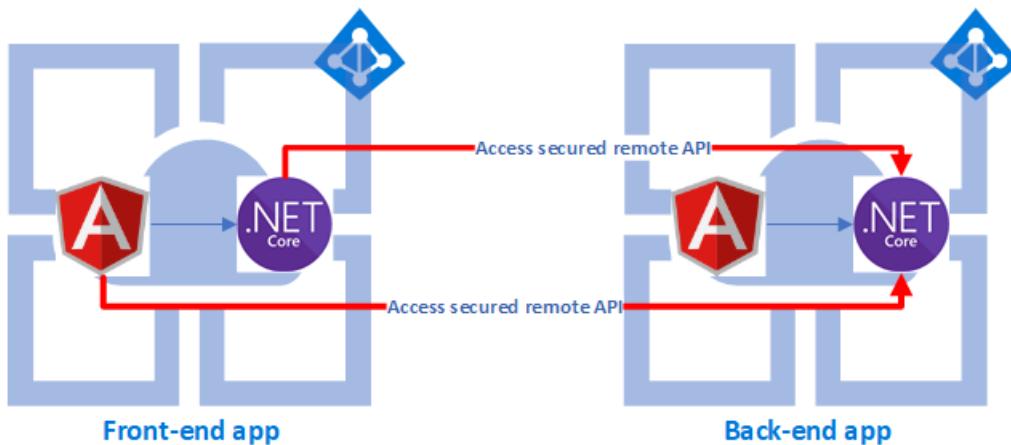
12/2/2019 • 13 minutes to read • [Edit Online](#)

App Service on Linux 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 Angular.js front end, but it is only for 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.

The tutorial uses the sample app to show you how to secure a self-contained app (in [Enable authentication and authorization for back-end app](#)).



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 .NET Core](#).

## Create local .NET Core app

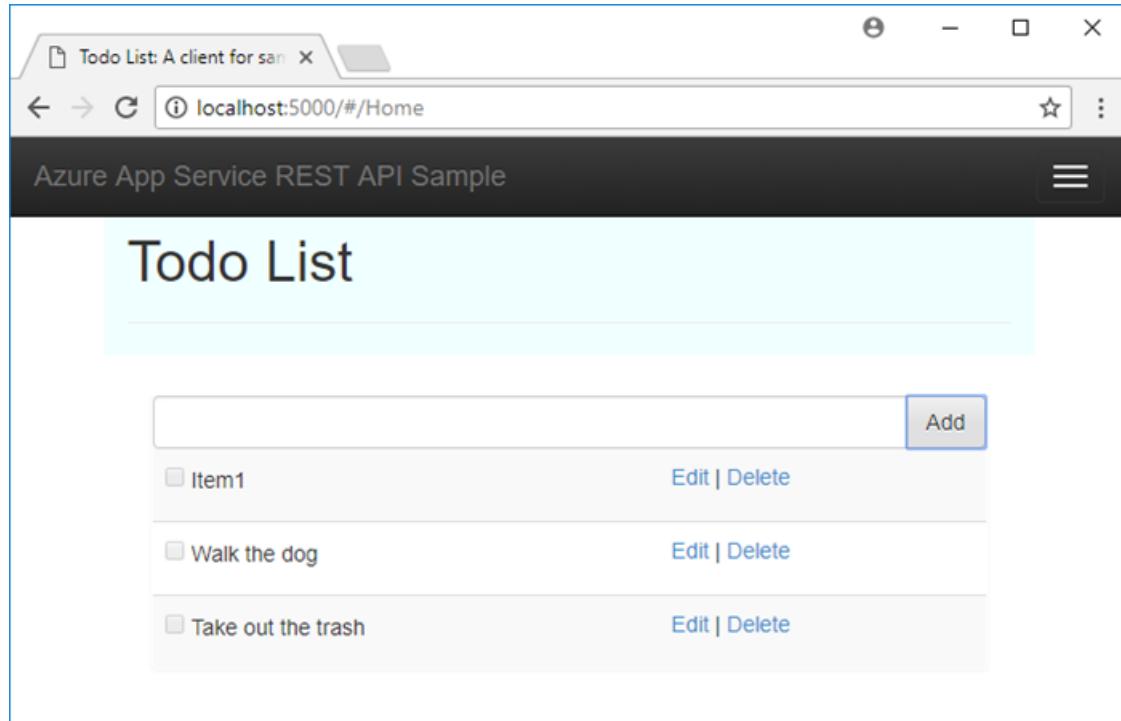
In this step, you set up the local .NET Core project. You use the same project to deploy a back-end API app and a front-end web app.

### Clone and run the sample application

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
```

Navigate to <http://localhost:5000> and try adding, editing, and removing todo items.



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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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 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.

### Create Azure resources

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 on Linux](#).

```
az group create --name myAuthResourceGroup --location "West Europe"
az appservice plan create --name myAuthAppServicePlan --resource-group myAuthResourceGroup --sku B1 --is-linux
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <front-end-app-name>
--runtime "dotnetcore|2.0" --deployment-local-git --query deploymentLocalGitUrl
az webapp create --resource-group myAuthResourceGroup --plan myAuthAppServicePlan --name <back-end-app-name> -
--runtime "dotnetcore|2.0" --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

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 master
```

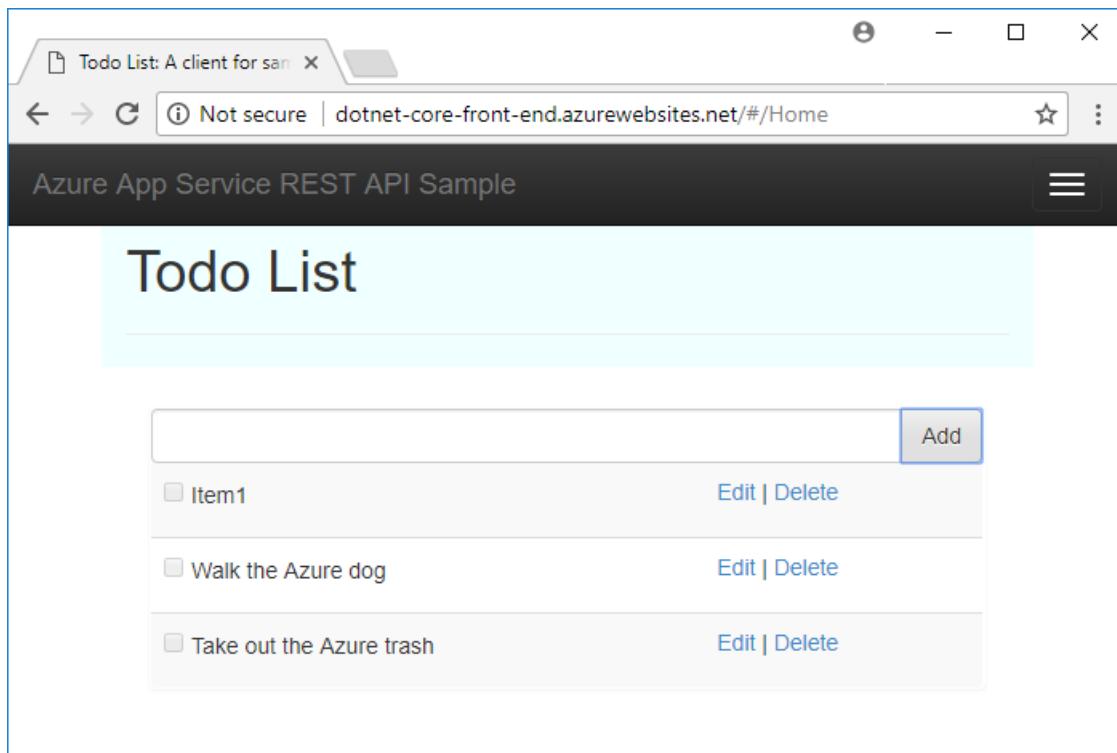
In the local terminal window, run the following Git commands to deploy the same code to the front-end app. Replace *<deploymentLocalGitUrl-of-front-end-app>* with the URL of the Git remote that you saved from [Create Azure resources](#).

```
git remote add frontend <deploymentLocalGitUrl-of-front-end-app>
git push frontend master
```

## 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

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";
```

Find the  `GetAll()` method and replace the code inside the curly braces with:

```
var data = _client.GetStringAsync($"{_remoteUrl}/api/Todo").Result;
return JsonConvert.DeserializeObject<List<TodoItem>>(data);
```

The first line makes a  `GET /api/Todo` call to the back-end API app.

Next, find the  `GetById(long id)` method and replace the code inside the curly braces with:

```
var data = _client.GetStringAsync($"{_remoteUrl}/api/Todo/{id}").Result;
return Content(data, "application/json");
```

The first line makes a  `GET /api/Todo/{id}` call to the back-end API app.

Next, find the  `Create([FromBody] TodoItem item)` method and replace the code inside the curly braces with:

```
var response = _client.PostAsJsonAsync($"{_remoteUrl}/api/Todo", item).Result;
var data = response.Content.ReadAsStringAsync().Result;
return Content(data, "application/json");
```

The first line makes a  `POST /api/Todo` call to the back-end API app.

Next, find the  `Update(long id, [FromBody] TodoItem item)` method and replace the code inside the curly braces with:

```
var res = _client.PutAsJsonAsync($"{_remoteUrl}/api/Todo/{id}", item).Result;
return new NoContentResult();
```

The first line makes a  `PUT /api/Todo/{id}` call to the back-end API app.

Next, find the  `Delete(long id)` method and replace the code inside the curly braces with:

```
var res = _client.DeleteAsync($"{_remoteUrl}/api/Todo/{id}").Result;
return new NoContentResult();
```

The first line makes a  `DELETE /api/Todo/{id}` call to the back-end API app.

Save your 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 master
```

## Check your changes

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`.

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.

A screenshot of a web browser window titled "Todo List: A client for san". The address bar shows "dotnet-core-front-end.azurewebsites.net/#/Home". The page content is titled "Azure App Service REST API Sample" and "Todo List". It lists five items:

- Item1 (checkbox checked)
- from front end 1
- from front end 2
- from back end 1
- from back end 2

Each item has "Edit | Delete" links to its right. An "Add" button is located at the top right of the list area.

## 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

In the [Azure portal](#), open your back-end app's management page by clicking from the left menu: **Resource groups > myAuthResourceGroup > <back-end-app-name>**.

A screenshot of the Azure portal showing the "Resource groups" blade. On the left sidebar, under "FAVORITES", the "Resource groups" option is selected and highlighted with a red box. In the main pane, the "myAuthResourceGroup" resource group is selected and highlighted with a red box. The "dotnet-core-back-end" app service is listed under the "myAuthResourceGroup" section and is also highlighted with a red box.

In your back-end app's left menu, click **Authentication / Authorization**, then enable App Service Authentication by clicking **On**.

In **Action to take when request is not authenticated**, select **Log in with Azure Active Directory**.

Under **Authentication Providers**, click **Azure Active Directory**

The screenshot shows the 'dotnet-core-back-end - Authentication / Authorization' settings in the Azure portal. The left sidebar has a tree view with 'Authentication / Authorization' selected. In the main pane, 'App Service Authentication' is set to 'On'. The 'Action to take when request is not authenticated' dropdown is set to 'Log in with Azure Active Directory'. The 'Authentication Providers' section shows 'Azure Active Directory' as 'Not Configured', while 'Microsoft' and 'Facebook' are also listed.

Click **Express**, then accept the default settings to create a new AD app and click **OK**.

In the **Authentication / Authorization** page, click **Save**.

Once you see the notification with the message `Successfully saved the Auth Settings for <back-end-app-name> App`, refresh the page.

Click **Azure Active Directory** again, and then click the **Azure AD App**.

Copy the **Client ID** of the Azure AD application to a notepad. You need this value later.

The screenshot shows the 'Azure Active Directory Settings' and 'Azure AD Applications' pages. On the left, under 'Active Directory Authentication', 'Management mode' is set to 'Express'. On the right, the 'Azure AD Applications' table lists a single application named 'dotnet-core-back-end' with its 'CLIENT ID' (d86ca7ce-55b2-495c-94f2-4b09b25b5164) highlighted.

### 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.

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 you still 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

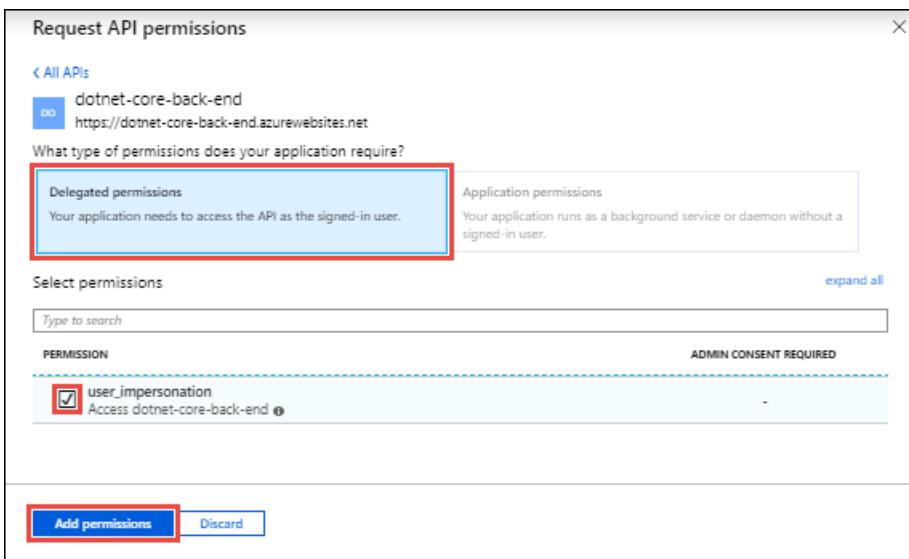
Now that you've enabled authentication and authorization to both of your apps, each of them is backed by an AD application. In this step, you give the front-end app permissions to access the back end on the user's behalf. (Technically, you give the front end's *AD application* the permissions to access the back end's *AD application* on the user's behalf.)

From the left menu in the portal, select **Azure Active Directory > App registrations > Owned applications > <front-end-app-name> > API permissions**.

The screenshot shows the Azure portal interface. On the left, the navigation menu is open, with the 'Azure Active Directory' section highlighted. In the center, the 'Microsoft - App registrations' blade is displayed for the 'dotnetcore-front-end' app. The 'Owned applications' tab is selected. A table lists two applications: 'dotnet-core-front-end' and 'dotnet-core-back-end'. On the right side of the blade, the 'API permissions' section is visible, with a red box highlighting the 'API permissions' link under the 'Manage' category. The URL for the page is: Home > Microsoft - App registrations > dotnetcore-front-end - API permissions.

Select **Add a permission**, then select **My APIs > <back-end-app-name>**.

In the **Request API permissions** page for the back-end app, select **Delegated permissions** and **user\_impersonation**, then select **Add permissions**.



## 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](#).

Sign in to [Azure Resource Explorer](#). At the top of the page, click **Read/Write** to enable editing of your Azure resources.



In the left browser, click **subscriptions** > **<your-subscription>** > **resourceGroups** > **myAuthResourceGroup** > **providers** > **Microsoft.Web** > **sites** > **<front-end-app-name>** > **config** > **authsettings**.

In the **authsettings** view, click **Edit**. Set `additionalLoginParams` to the following JSON string, using the client ID you copied.

```
"additionalLoginParams": ["response_type=code id_token", "resource=<back-end-client-id>"],  
  
15  "allowedAudiences": [  
16    "https://dotnet-core-front-end.azurewebsites.net/.auth/login/aad/callback",  
17    "(String)"  
18  ],  
19  "additionalLoginParams": ["response_type=code id_token", "resource=ad402dfa-0b9c-4b61-a584-a6a503597fa9"],  
20  "isAADAutoProvisioned": true,  
21  "aadClientID": null
```

Save your settings by clicking **PUT**.

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.

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 method (such as `GetAll()`) does, so each of your outgoing API call now presents the access token.

Save your 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 master
```

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 Angular.js 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 [Active Directory Authentication Library \(ADAL\) for JavaScript](#) to help simplify the Angular.js application pattern.

## Configure CORS

In the Cloud Shell, enable CORS to your client's URL by using the `az resource update` command. Replace the `<back-end-app-name>` and `<front-end-app-name>` placeholders.

```
az resource update --name web --resource-group myAuthResourceGroup --namespace Microsoft.Web --resource-type config --parent sites/<back-end-app-name> --set properties.cors.allowedOrigins="['https://<front-end-app-name>.azurewebsites.net']" --api-version 2015-06-01
```

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 Angular.js app. For more information, see [Add CORS functionality](#).

## Point Angular.js app to back-end API

In the local repository, open `wwwroot/index.html`.

In Line 51, set the `apiEndpoint` variable to the 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.

In the local repository, open `wwwroot/app/scripts/todoListSvc.js` and see that `apiEndpoint` is prepended to all the API calls. Your Angular.js app is now calling the back-end APIs.

## Add access token to API calls

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.

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" });
```

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 `revolve` 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

Save your 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 master
```

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](#)

# CLI samples for Azure App Service

12/10/2019 • 2 minutes to read • [Edit Online](#)

The following table includes links to bash scripts built using the Azure CLI.

<b>Create app</b>	
<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.
<b>Configure app</b>	
<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 SSL certificate to an app</a>	Creates an App Service app and binds the SSL certificate of a custom domain name to it.
<b>Scale app</b>	
<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.
<b>Protect app</b>	
<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.
<b>Connect app to resources</b>	
<a href="#">Connect an app to a SQL Database</a>	Creates an App Service app and a 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>Back up and restore app</b>	
<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="#">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

12/2/2019 • 2 minutes to read • [Edit Online](#)

The following table includes links to PowerShell scripts built using the Azure PowerShell.

<b>Create app</b>	
<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.
<b>Configure app</b>	
<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 SSL certificate to an app</a>	Creates an App Service app and binds the SSL certificate of a custom domain name to it.
<b>Scale app</b>	
<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.
<b>Connect app to resources</b>	
<a href="#">Connect an app to a SQL Database</a>	Creates an App Service app and a 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.
<b>Back up and restore app</b>	
<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 App Service plan overview

2/20/2020 • 7 minutes to read • [Edit Online](#)

In App Service, an app runs 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:

- 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, Isolated)

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

- **Shared compute:** **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**, and **PremiumV2** 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 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 **PremiumV2** pricing tier provides [Dv2-series VMs](#) with faster processors, SSD storage, and double memory-to-core ratio compared to **Standard** tier. **PremiumV2** also supports higher scale via increased instance count while still providing all the advanced capabilities found in the Standard plan. All features available in the existing **Premium** tier are included in **PremiumV2**.

Similar to other dedicated tiers, three VM sizes are available for this tier:

- Small (one CPU core, 3.5 GiB of memory)
- Medium (two CPU cores, 7 GiB of memory)
- Large (four CPU cores, 14 GiB of memory)

For **PremiumV2** pricing information, see [App Service Pricing](#).

To get started with the new **PremiumV2** pricing tier, see [Configure PremiumV2 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 an hourly charge on the compute resources it uses.

- In the **Shared** tier, each app receives a quota of CPU minutes, so *each app* is charged hourly for the CPU quota.
- In the dedicated compute tiers (**Basic**, **Standard**, **Premium**, **PremiumV2**), the App Service plan defines the number of VM instances the apps are scaled to, so *each VM instance* in the App Service plan has an hourly charge. 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 hourly. In addition, there's an hourly base 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, 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 SSL connections - There's an hourly charge for each IP-based SSL connection, but some **Standard** tier or above gives you one IP-based SSL connection for free. SNI-based SSL 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](#).

## 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 an SSL 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](#)

# Introduction to the App Service Environments

1/8/2020 • 4 minutes to read • [Edit Online](#)

## Overview

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
- Mobile apps
- Functions

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

- Very high scale.
- Isolation and secure network access.
- High memory utilization.

Customers can create multiple ASEs within a single Azure region or across multiple Azure regions. This flexibility makes ASEs ideal for horizontally scaling stateless application tiers in support of high RPS workloads.

ASEs are isolated to running only a single customer's applications and are always deployed into a virtual network. 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.

- ASE comes with its own pricing tier, learn how the [Isolated offering](#) helps drive hyper-scale and security.
- [App Service Environments v2](#) provide a surrounding to safeguard your apps in a subnet of your network and provides your own private deployment of Azure App Service.
- Multiple ASEs can be used to scale horizontally. For more information, see [how to set up a geo-distributed app footprint](#).
- ASEs can be used to configure security architecture, as shown in the AzureCon Deep Dive. To see how the security architecture shown in the AzureCon Deep Dive was configured, see the [article on how to implement a layered security architecture](#) with App Service environments.
- Apps running on ASEs can have their access gated by upstream devices, such as web application firewalls (WAFs). For more information, see [Web application firewall \(WAF\)](#).
- App Service Environments can be deployed into Availability Zones (AZ) using zone pinning. See [App Service Environment Support for Availability Zones](#) for more details.

## Dedicated environment

An ASE is dedicated exclusively to a single subscription and can host 100 App Service Plan instances. The range can span 100 instances in a single App Service plan to 100 single-instance App Service plans, and everything in between.

An ASE is composed of front ends and workers. Front ends are responsible for HTTP/HTTPS termination and automatic load balancing of app requests within an ASE. Front ends are automatically added as the App Service

plans in the ASE are scaled out.

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

Customers do not need to manage front ends and workers. All infrastructure is automatically added as customers scale out their App Service plans. As App Service plans are created or scaled in an ASE, the required infrastructure is added or removed as appropriate.

There is a flat monthly rate for an ASE that pays for the infrastructure and doesn't change with the size of the ASE. In addition, there is a cost per App Service plan vCPU. All apps hosted in an ASE are in the Isolated pricing SKU. For information on pricing for an ASE, see the [App Service pricing](#) page and review the available options for ASEs.

## Virtual network support

The ASE feature is a deployment of the Azure App Service directly into a customer's Azure Resource Manager virtual network. To learn more about Azure virtual networks, see the [Azure virtual networks FAQ](#). An ASE always exists in a virtual network, and more precisely, within a subnet of a virtual network. You can use the security features of virtual networks to control inbound and outbound network communications for your apps.

An ASE can be either internet-facing with a public IP address or internal-facing with only an Azure internal load balancer (ILB) address.

[Network Security Groups](#) restrict inbound network communications to the subnet where an ASE resides. You can use NSGs to run apps behind upstream devices and services such as WAFs and network SaaS providers.

Apps also frequently need to access corporate resources such as internal databases and web services. If you deploy the ASE in a virtual network that has a VPN connection to the on-premises network, the apps in the ASE can access the on-premises resources. This capability is true regardless of whether the VPN is a [site-to-site](#) or [Azure ExpressRoute](#) VPN.

For more information on how ASEs work with virtual networks and on-premises networks, see [App Service Environment network considerations](#).

## App Service Environment v1

App Service Environment has two versions: ASEv1 and ASEv2. The preceding information was based on ASEv2. This section shows you the differences between ASEv1 and ASEv2.

In ASEv1, you need to manage all of the resources manually. That includes the front ends, workers, and IP addresses used for IP-based SSL. Before you can scale out your App Service plan, you need to first scale out the worker pool where you want to host it.

ASEv1 uses a different pricing model from ASEv2. In ASEv1, you pay for each vCPU allocated. That includes vCPUs used for front ends or workers that aren't hosting any workloads. In ASEv1, the default maximum-scale size of an ASE is 55 total hosts. That includes workers and front ends. One advantage to ASEv1 is that it can be deployed in a classic virtual network and a Resource Manager virtual network. To learn more about ASEv1, see [App Service Environment v1 introduction](#).

# Inbound and outbound IP addresses in Azure App Service

12/2/2019 • 2 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.

## 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.
- Delete the last app in a resource group *and* region combination and recreate it.
- Delete an existing 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 configure an [IP-based SSL binding](#). If you don't actually need SSL functionality to secure your app, you can even upload a self-signed certificate for this binding. In an IP-based SSL 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. You can't know beforehand which IP address a given app instance will use to make the outbound connection, so your back-end service must open its firewall to all the outbound IP addresses of your app.

The set of outbound IP addresses for your app changes when you scale your app between the lower tiers (**Basic**, **Standard**, and **Premium**) and the **Premium V2** tier.

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
```

## Next steps

Learn how to restrict inbound traffic by source IP addresses.

[Static IP restrictions](#)

# Authentication and authorization in Azure App Service

1/23/2020 • 7 minutes to read • [Edit Online](#)

## NOTE

At this time, AAD V2 (including MSAL) is not supported for Azure App Services and Azure Functions. Please check back for updates.

Azure App Service provides built-in authentication and authorization support, so you can sign in users and access data by writing minimal or no code in your web app, RESTful API, and mobile back end, and also [Azure Functions](#). This article describes how App Service helps simplify authentication and authorization for your app.

Secure authentication and authorization require deep understanding of security, including federation, encryption, [JSON web tokens \(JWT\)](#) management, [grant types](#), and so on. App Service provides these utilities so that you can spend more time and energy on providing business value to your customer.

## IMPORTANT

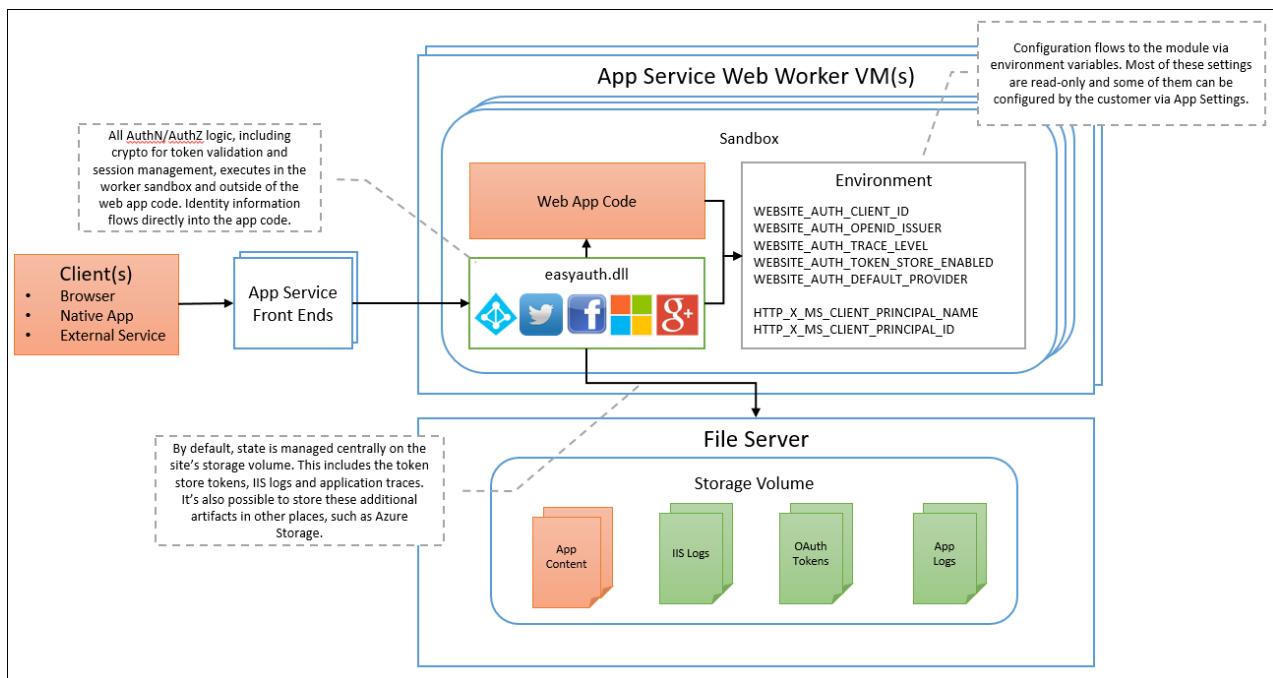
You're not required to use App Service for AuthN/AuthO. You can use the bundled security features in your web framework of choice, or you can write your own utilities. However, keep in mind that [Chrome 80 is making breaking changes to its implementation of SameSite for cookies](#) (release date around March 2020), and custom remote authentication or other scenarios that rely on cross-site cookie posting may break when client Chrome browsers are updated. The workaround is complex because it needs to support different SameSite behaviors for different browsers.

The ASP.NET Core 2.1 and above versions hosted by App Service are already patched for this breaking change and handle Chrome 80 and older browsers appropriately. In addition, the same patch for ASP.NET Framework 4.7.2 is being deployed on the App Service instances throughout January 2020. For more information, including how to know if your app has received the patch, see [Azure App Service SameSite cookie update](#).

For information specific to native mobile apps, see [User authentication and authorization for mobile apps with Azure App Service](#).

## How it works

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



This module handles several things for your app:

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

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

### User claims

For all language frameworks, App Service makes the user's claims available to your code by injecting them into the request headers. 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 hydrated for .NET code, but you can still find the user claims in the request headers.

For more information, see [Access user claims](#).

### 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 from the Azure Active Directory Graph API or even the Microsoft Graph

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 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.

## 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`.

## Identity providers

App Service uses [federated identity](#), in which a third-party identity provider manages the user identities and authentication flow for you. Five identity providers are available by default:

Provider	Sign-in endpoint
Azure Active Directory	<code>/auth/login/aad</code>
Microsoft Account	<code>/auth/login/microsoftaccount</code>
Facebook	<code>/auth/login/facebook</code>
Google	<code>/auth/login/google</code>
Twitter	<code>/auth/login/twitter</code>

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 with ease. You can also integrate another identity provider or [your own custom identity solution](#).

## 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.

### Note

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 authentication and authorization in App Service](#).

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.
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 authorization with a number of behaviors when incoming request is not authenticated.



The following headings describe the options.

### Allow Anonymous requests (no action)

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.

### Allow only authenticated requests

The option is **Log in with <provider>**. App Service redirects all anonymous requests 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`.

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.

## More resources

[Tutorial: Authenticate and authorize users end-to-end in Azure App Service \(Windows\)](#)

[Tutorial: Authenticate and authorize users end-to-end in Azure App Service for Linux](#)

[Customize authentication and authorization in App Service](#)

Provider-specific how-to guides:

- [How to configure your app to use Azure Active Directory login](#)
- [How to configure your app to use Facebook login](#)
- [How to configure your app to use Google login](#)
- [How to configure your app to use Microsoft Account login](#)
- [How to configure your app to use Twitter login](#)
- [How to: Use custom authentication for your application](#)

# Custom image, multi-container, or built-in platform image?

12/2/2019 • 3 minutes to read • [Edit Online](#)

[App Service on Linux](#) offers three different paths to getting your application published to the web:

- **Custom image deployment:** "Dockerize" your app into a Docker image that contains all of your files and dependencies in a ready-to-run package.
- **Multi-container deployment:** "Dockerize" your app across multiple containers using a Docker Compose configuration file.
- **App deployment with a built-in platform image:** Our built-in platform images contain common web app runtimes and dependencies, such as Node and PHP. Use any one of the [Azure App Service deployment methods](#) to deploy your app to your web app's storage, and then use a built-in platform image to run it.

## Which method is right for your app?

The primary factors to consider are:

- **Availability of Docker in your development workflow:** Custom image development requires basic knowledge of the Docker development workflow. Deployment of a custom image to a web app requires publication of your custom image to a repository host like Docker Hub. If you are familiar with Docker and can add Docker tasks to your build workflow, or if you are already publishing your app as a Docker image, a custom image is almost certainly the best choice.
- **Multi-layered architecture:** Deploy multiple containers such as a web application layer and an API layer to separate capabilities by using multi-container.
- **Application performance:** Increase the performance of your multi-container app using a cache layer such as Redis. Select multi-container to achieve this.
- **Unique runtime requirements:** The built-in platform images are designed to meet the needs of most web apps, but are limited in their customizability. Your app may have unique dependencies or other runtime requirements that exceed what the built-in images are capable of.
- **Build requirements:** With [continuous deployment](#), you can get your app up and running on Azure directly from source code. No external build or publication process is required. However, there is a limit to the customizability and availability of build tools within the [Kudu](#) deployment engine. Your app may outgrow Kudu's capabilities as it grows in its dependencies or requirements for custom build logic.
- **Disk read/write requirements:** All web apps are allocated a storage volume for web content. This volume, backed by Azure Storage, is mounted to `/home` in the app's filesystem. Unlike files in the container filesystem, files in the content volume are accessible across all scale instances of an app, and modifications will persist across app restarts. However, the disk latency of the content volume is higher and more variable than the latency of the local container filesystem, and access can be impacted by platform upgrades, unplanned downtime, and network connectivity issues. Apps that require heavy read-only access to content files may benefit from custom image deployment, which places files in the image filesystem instead of on the content volume.
- **Build resource usage:** When an app is deployed from source, the deployment scripts run by Kudu use the same App Service Plan compute and storage resources as the running app. Large app deployments may consume more resources or time than desired. In particular, many deployment workflows generate heavy disk activity on the app content volume, which is not optimized for such activity. A custom image delivers all of your app's files and dependencies to Azure in a single package with no need for additional file transfers or deployment actions.

- **Need for rapid iteration:** Dockerizing an app requires additional build steps. For changes to take effect, you must push your new image to a repository with each update. These updates are then pulled to the Azure environment. If one of the built-in containers meets your app's needs, deploying from source may offer a faster development workflow.

## Next steps

Custom container:

- [Run custom container](#)

Multi-container:

- [Create multi-container app](#)

The following articles get you started with App Service on Linux with a built-in platform image:

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

# Azure App Service diagnostics overview

12/2/2019 • 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** for Windows apps. See [Diagnostic tools \(only for Windows app\)](#).

The screenshot shows the Microsoft Azure portal interface. The left sidebar is visible with various service icons. The main content area is titled 'BuggyBakery' and shows the 'App Service Diagnostics' section. It features several tiles:

- Availability and Performance**: Describes downtime or slowness and includes keywords like Health Check, Downtime, 5xx Errors, 4xx Errors, CPU, and Memory.
- Configuration and Management**: Describes issues with configuration and includes keywords like Scaling, Swaps, Failed Backups, IPs, Migration, and 4xx Errors.
- SSL and Domains**: Describes SSL certificate and domain management issues and includes keywords like 4xx Errors, SSL, Domains, Permissions, Auth, and Cert.
- Best Practices**: Describes running applications in production and includes keywords like AutoScale, Traffic Manager, AlwaysOn, ARR Affinity, Metrics, and Security.
- Diagnostic Tools**: Describes deeper investigation and includes keywords like Profiler, Memory Dump, DnsS, AutoHeal, Metrics, and Security.

## 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 top 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.' A grid of nine blue tiles provides quick access to diagnostic reports:

- 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 'Availability and Performance' tab selected. 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.' A button labeled 'I am interested in Application Logs' is visible. A message below 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.' A list of topics is shown, with the first item highlighted by a red border:

- Wrong port is exposed: 5000 != 8080
- Verbose application logging is off.
- Link to Full Logs

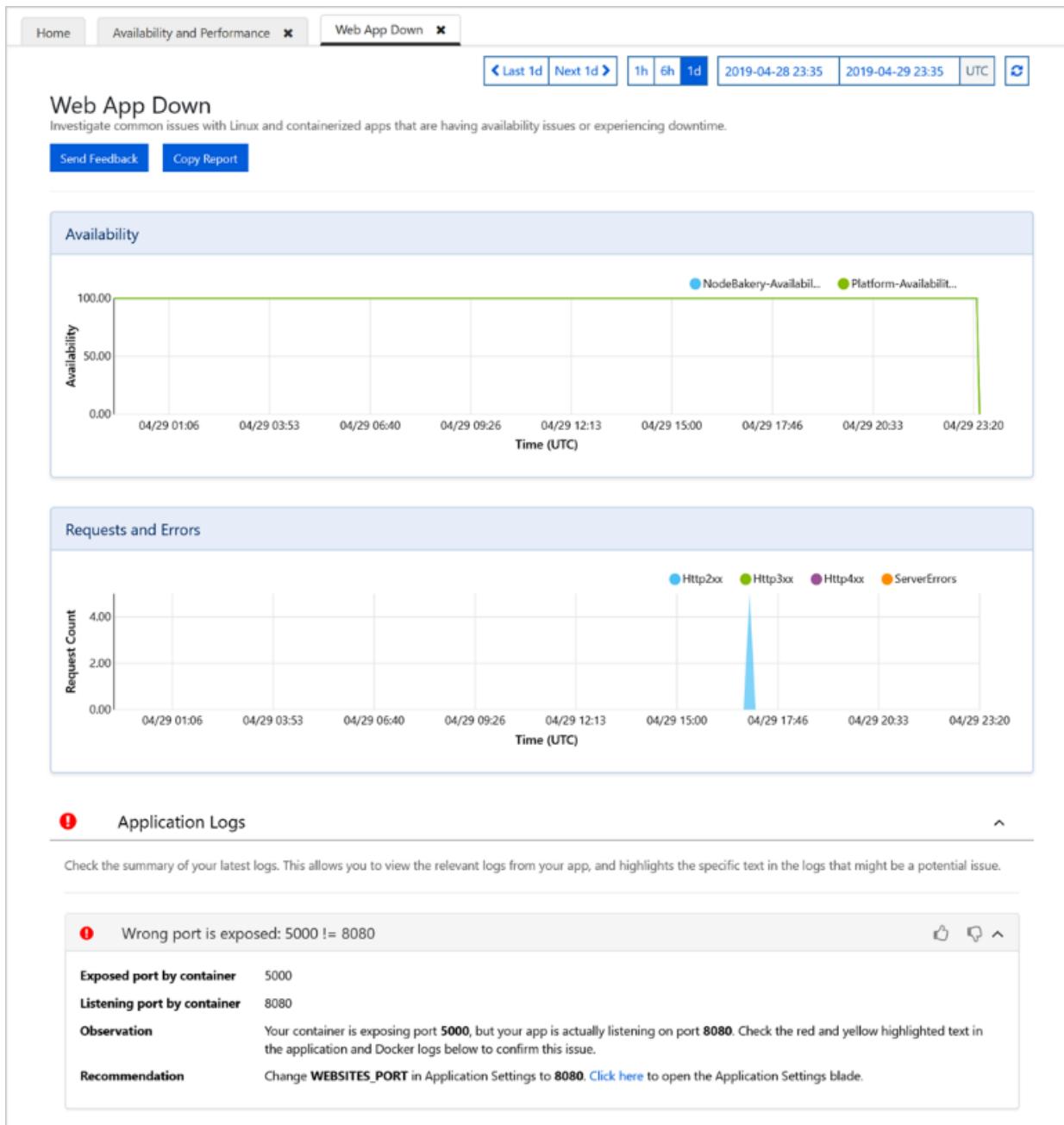
Web App Down

[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.



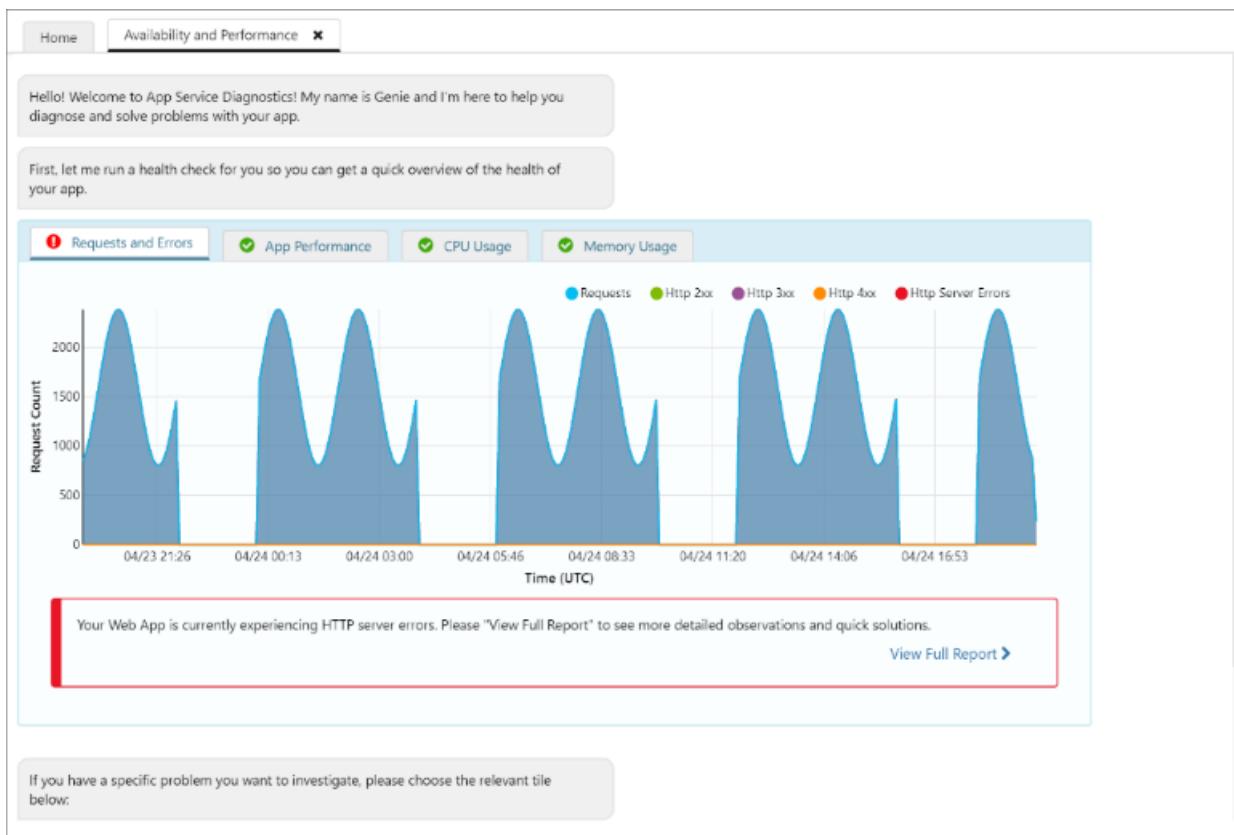
## Health checkup

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.



If you have a specific problem you want to investigate, please choose the relevant tile below:

## 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**

<b>S1 Standard</b>	<b>1</b>	<b>2</b>
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 (only for Windows app)

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

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



75% 95%

**Threshold Seconds**  
For the rule to trigger, CPU should exceed 75% for this many seconds



30 sec 180 sec

## Auto-healing and proactive 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. For more information, see [Announcing the new auto healing experience in app service diagnostics](#).

Configure Mitigation Rules    ProActive Auto-Healing    View Auto-Healing History

Auto-Healing Enabled    Off    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

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				
July 2019	Thu 1 August 2019	●	●	Fri 2 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.N 4   <a href="https://go.microsoft.com/fwlink/?LinkId=301879">https://go.microsoft.com/fwlink/?LinkId=301879</a> 5   --&gt; 6 &lt;configuration&gt; 7   &lt;connectionStrings&gt; 8 -   &lt;add name="MyDbConnection" connectionString="Server=(local);Datab 9   &lt;add name="StorageConnection" connectionString="Data Source=(local);D 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> <pre> 1 &lt;?xml version="1.0" encoding="utf-8"?&gt; 2 &lt;!-- 3   For more information on how to configure your ASP.N 4   <a href="https://go.microsoft.com/fwlink/?LinkId=301879">https://go.microsoft.com/fwlink/?LinkId=301879</a> 5   --&gt; 6 &lt;configuration&gt; 7   &lt;connectionStrings&gt; 8 +   &lt;add name="MyDbConnection" connectionString="Server=(local);Datab 9   &lt;add name="StorageConnection" connectionString="Data Source=(local);D 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

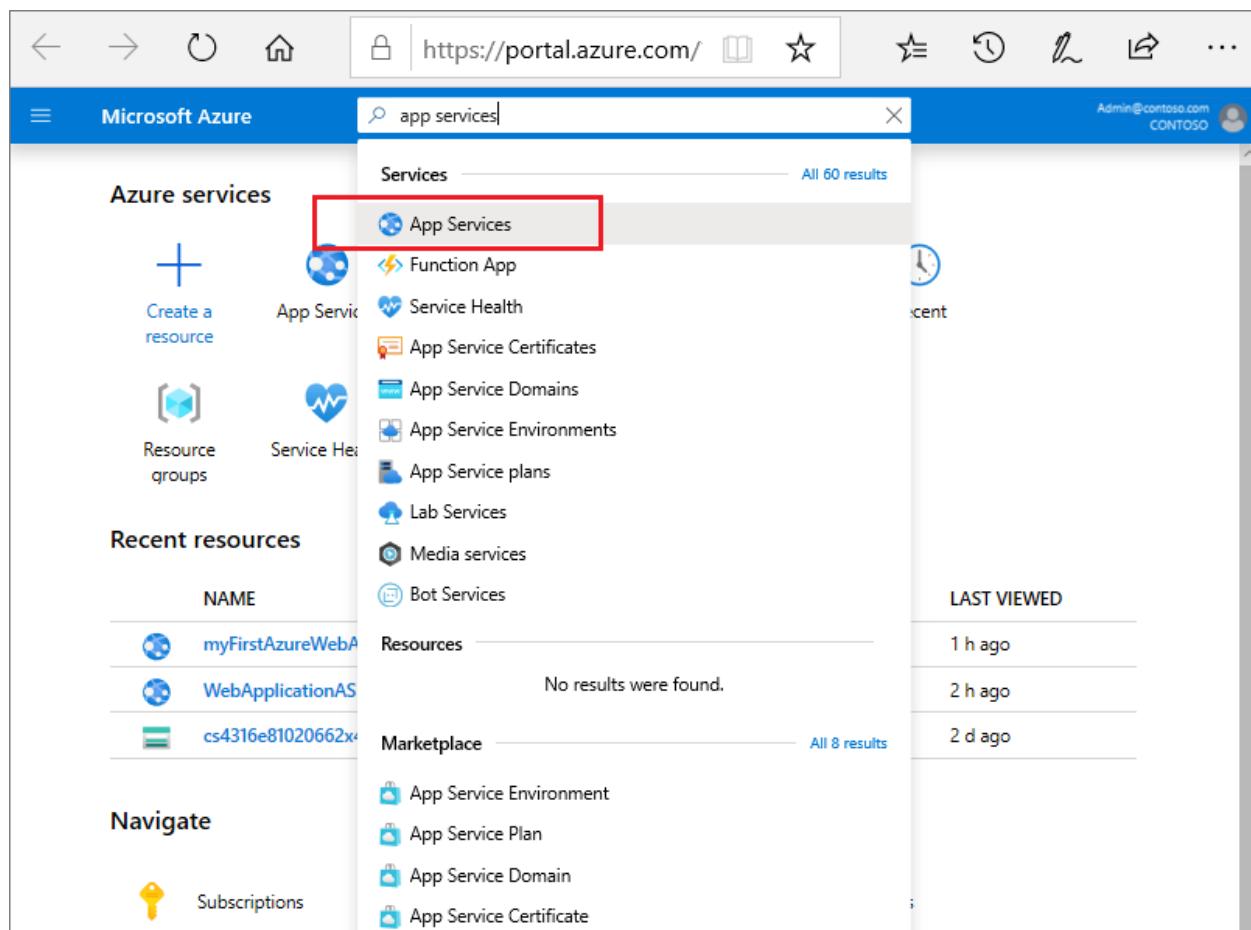
2/25/2020 • 8 minutes to read • [Edit Online](#)

This topic 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 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, home, and search. The search bar contains the text "app services". Below the search bar, the main content area has a sidebar on the left with sections for "Azure services", "Recent resources", and "Navigate". The "Azure services" section includes links for "Create a resource", "Resource groups", and "Service Health". The "Recent resources" section lists three items: "myFirstAzureWebA", "WebApplicationAS", and "cs4316e81020662x". The "Navigate" section includes a "Subscriptions" link. The main content area is titled "Services" and shows a list of items under "All 60 results". The item "App Services" 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". To the right of the list, there's a "LAST VIEWED" section showing "1 h ago", "2 h ago", and "2 d ago". At the bottom right, there's a vertical ellipsis icon.

In the app's left menu, select **Configuration > Application settings**.

The screenshot shows the Azure portal's Configuration page for an App Service. The left sidebar lists various settings like Security, Deployment, and Settings, with 'Configuration' selected. The top navigation bar has tabs for Application settings, General settings, Default documents, and Path mappings, with 'Application settings' highlighted. Below the tabs, there are sections for Application settings and Connection strings, each with a table for managing values. Both sections show '(no application settings to display)' and '(no connection strings to display)'.

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*, but production secrets (for example, Azure MySQL database password) safe 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 container 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  
  },  
  ...  
]
```

## 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**.

The screenshot shows the Azure portal's Configuration blade for an app named 'my-core-app'. The left sidebar lists various configuration sections: Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings (Configuration, 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 'Configuration' section is currently selected and highlighted with a red box. The main content area has tabs for Application settings, General settings, Default documents, and Path mappings. The 'Application settings' tab is active and highlighted with a red box. It contains a note about encrypted application settings and a table with one row: 'Name' (no application settings to display). Below this, the 'Connection strings' section is shown with a note about encrypted connection strings and a table with one row: 'Name' (no connection strings to display). At the top of the blade, there are Save and Discard buttons.

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. Here's one exception, however: certain Azure database types are backed up along with the app if you configure their connection strings in your 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:

- SQL Server: `SQLCONNSTR_`
- MySQL: `MYSQLCONNSTR_`
- SQL Database: `SQLAZURECONNSTR_`
- Custom: `CUSTOMCONNSTR_`

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**.

The screenshot shows the 'my-core-app - Configuration' blade in the Azure portal. The left sidebar lists various configuration sections: Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings (Configuration, 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 'Configuration' section is highlighted with a red box. The main area shows the 'General settings' tab selected (also highlighted with a red box). It includes sections for Application settings and Connection strings, both of which currently have no items displayed.

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 also set an optional start-up command or file.
- **Platform settings:** Lets you configure settings for the hosting platform, including:
  - **Bitness:** 32-bit or 64-bit.
  - **WebSocket protocol:** For [ASP.NET SignalR](#) or [socket.io](#), for example.
  - **Always On:** Keep the app loaded even when there's no traffic. It's required for continuous WebJobs or for WebJobs that are triggered using a CRON expression.

### NOTE

With the Always On feature, you can't control the endpoint. It always sends a request to the application root.

- **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

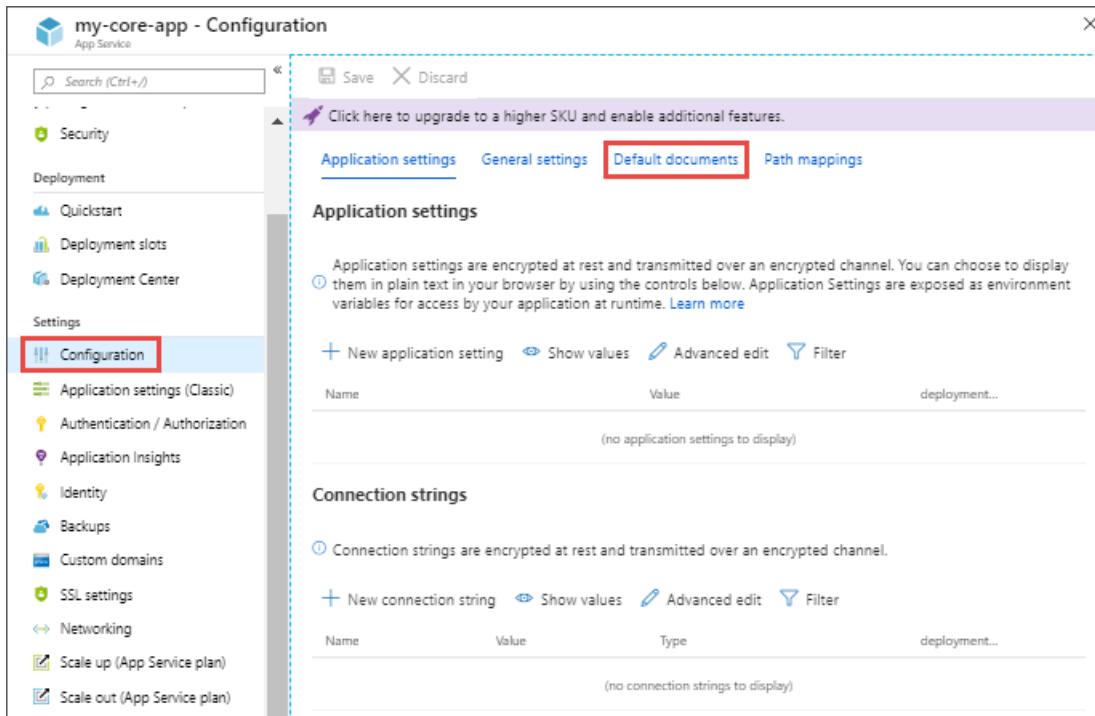
Most modern browsers support HTTP/2 protocol over TLS only, while non-encrypted traffic continues to use HTTP/1.1. To ensure that client browsers connect to your app with HTTP/2, [secure your custom DNS name with an SSL binding in Azure App Service](#).

- **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 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 interface for an app service named 'my-core-app'. The left sidebar lists various settings like Security, Deployment, and Configuration (which is selected and highlighted with a red box). The main content area has tabs for Application settings, General settings, Default documents, and Path mappings (also highlighted with a red box). Under Application settings, it says 'Application settings are encrypted at rest and transmitted over an encrypted channel. You can choose to display them in plain text in your browser by using the controls below. Application Settings are exposed as environment variables for access by your application at runtime.' Below this are buttons for New application setting, Show values, Advanced edit, and Filter. A table header shows columns for Name, Value, and deployment..., with a note '(no application settings to display)'. Under Connection strings, it says 'Connection strings are encrypted at rest and transmitted over an encrypted channel.' Below this are buttons for New connection string, Show values, Advanced edit, and Filter. A table header shows columns for Name, Value, Type, and deployment..., with a note '(no connection strings to display)'.

The **Path mappings** page shows you different things based on the OS type.

### 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**. Configure the handler as follows:

- **Extension.** The file extension you want to handle, such as `*.php` or `handlerfcgi`.
- **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. Click **New virtual application or directory**.

To configure virtual applications and directories, specify each virtual directory and its corresponding physical path relative to the website root (`D:\home`). Optionally, you can select the **Application** checkbox to mark a virtual directory as an application.

### 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 [Serve content from Azure Storage in App Service on Linux](#).

## Configure language stack settings

For Linux apps, see:

- [ASP.NET Core](#)
- [Node.js](#)
- [PHP](#)
- [Python](#)
- [Java](#)
- [Ruby](#)

## Configure custom containers

See [Configure a custom Linux container for Azure App Service](#)

## Next steps

- [Configure a custom domain name in Azure App Service](#)
- [Set up staging environments in Azure App Service](#)
- [Secure a custom DNS name with an 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 a Linux ASP.NET Core app for Azure App Service

2/28/2020 • 5 minutes to read • [Edit Online](#)

ASP.NET Core apps must be deployed 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 who use a built-in Linux container in App Service. If you've never used Azure App Service, follow the [ASP.NET Core quickstart](#) and [ASP.NET Core with SQL Database tutorial](#) first.

## 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

Run the following command in the [Cloud Shell](#) to set the .NET Core version to 2.1:

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --linux-fx-version "DOTNETCORE|2.1"
```

## Customize build automation

If you deploy your app using Git or zip packages with build automation turned on, 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 App Service app setting  
            var myAppSetting = _configuration["MySetting"];  
            // 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 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.

## 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 **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, [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 middlewares.

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;
        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](#).

## Deploy multi-project solutions

When you deploy an ASP.NET repository to the deployment engine with a `.csproj` file in the root directory, the engine deploys the project. When you deploy an ASP.NET repository with a `.sln` file in the root directory, the engine picks the first Web Site or Web Application Project it finds as the App Service app. It's possible for the engine not to pick the project you want.

To deploy a multi-project solution, you can specify the project to use in App Service in two different ways:

### Using .deployment file

Add a `.deployment` file to the repository root and add the following code:

```
[config]
project = <project-name>/<project-name>.csproj
```

### Using app settings

In the [Azure Cloud Shell](#), add an app setting to your App Service app by running the following CLI command. Replace `<app-name>`, `<resource-group-name>`, and `<project-name>` with the appropriate values.

```
az webapp config appsettings set --name <app-name> --resource-group <resource-group-name> --settings PROJECT="<project-name>/<project-name>.csproj"
```

## Access diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

**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.

The screenshot shows a browser window with the URL `https://<app-name>.scm.azurewebsites.net/webssh/host`. The title bar says "SSH". The terminal window displays the output of the command `service --status-all`, listing numerous services like apache2, bootlogs, bootmisc.sh, checkfs.sh, etc. at the root level. The status for most services is marked with a minus sign (-), indicating they are not running.

```
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
[ - ]  rmnologin
[ - ]  sendsigs
[ + ]  ssh
[ + ]  udev
[ ? ]  udev-finish
[ - ]  umountfs
[ - ]  umountnfs.sh
[ - ]  umountroot
[ - ]  urandom
root@9e933156516f:~#
```

#### 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](#).

## Next steps

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

[App Service Linux FAQ](#)

# Configure a Linux Node.js app for Azure App Service

2/28/2020 • 7 minutes to read • [Edit Online](#)

Node.js apps must be deployed with all the required NPM dependencies. The App Service deployment engine (Kudu) automatically runs `npm install --production` for you when you deploy a [Git repository](#), or a [Zip package](#) with build processes switched on. 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 use a built-in Linux container in 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 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](#):

```
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.

## Customize build automation

If you deploy your app using Git or zip packages with build automation turned on, 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 Node.js 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.

- [Run custom command](#)
- [Run npm start](#)
- [Run with PM2](#)

### 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"
```

### 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>"
```

## 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, Kudu runs `npm install --production` when it recognizes a Node.js app is deployed. 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, [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

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

### 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.

```
root@9e933156516f:~# service --status-all
[ + ]  apache2
[ - ]  bootlogs
[ - ]  bootmisc.sh
[ - ]  checkfs.sh
[ - ]  checkroot-bootclean.sh
[ - ]  checkroot.sh
[ - ]  hostname.sh
[ ? ]  hwclock.sh
[ - ]  killprocs
[ - ]  mtd
[ - ]  mountall-bootclean.sh
[ - ]  mountall.sh
[ - ]  mountdevsubfs.sh
[ - ]  mountkernfs.sh
[ - ]  mountnfs-bootclean.sh
[ - ]  mountnfs.sh
[ - ]  mysql
[ - ]  procps
[ - ]  rc.local
[ - ]  rmnologin
[ - ]  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](#).

## 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](#).

## Next steps

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

[App Service Linux FAQ](#)

# Configure a Linux PHP app for Azure App Service

2/28/2020 • 7 minutes to read • [Edit Online](#)

This guide shows you how to configure the built-in PHP runtime for web apps, mobile back ends, and API apps in Azure App Service.

This guide provides key concepts and instructions for PHP developers who use a built-in Linux container in 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 linuxFxVersion
```

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 7.2:

```
az webapp config set --name <app-name> --resource-group <resource-group-name> --linux-fx-version "PHP|7.2"
```

## Customize build automation

If you deploy your app using Git or zip packages with build automation turned on, 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.

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

    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, [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['X-Forwarded-Proto']) && $_SERVER['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 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="/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 `.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

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

### 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.

```
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
[ - ] mountdebsubfs.sh
[ - ] mountkernfs.sh
[ - ] mountnfs-bootclean.sh
[ - ] mountnfs.sh
[ - ] mysql
[ - ] procps
[ - ] rc.local
[ - ] rmnologin
[ - ] 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](#).

## 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 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 `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

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](#)

# Configure a Linux Java app for Azure App Service

2/18/2020 • 22 minutes to read • [Edit Online](#)

Azure App Service on Linux lets Java developers quickly build, deploy, and scale their Tomcat, or Java Standard Edition (SE) packaged web applications on a fully managed Linux-based 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 who use a built-in Linux container in App Service. If you've never used Azure App Service, follow the [Java quickstart](#).

## Deploying your app

You can use [Maven Plugin for Azure App Service](#) to deploy both .jar and .war files. Deployment with popular IDEs is also supported with [Azure Toolkit for IntelliJ](#) or [Azure Toolkit for Eclipse](#).

Otherwise, your deployment method will depend on your archive type:

- To deploy .war files to Tomcat, use the `/api/wardeploy/` endpoint to POST your archive file. For more information on this API, please see [this documentation](#).
- To deploy .jar files on the Java SE images, use the `/api/zipdeploy/` endpoint of the Kudu site. For more information on this API, please see [this documentation](#).

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](#).

### 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.

```
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
[ - ]  rmmnlogin
[ - ]  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](#).

#### Stream diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

#### 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, see [Stream logs in Cloud Shell](#).

#### 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.

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](#).

## 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 Linux Java images on App Service have Zulu Flight Recorder installed so you can easily connect to the JVM and start a profiler recording or generate a heap dump.

#### Timed Recording

To get started, SSH into your App Service and run the `jcmd` command to see a list of all the Java processes running. In addition to `jcmd` itself, you should see your Java application running with a process ID number (pid).

```
078990bbcd11:/home# jcmd
Picked up JAVA_TOOL_OPTIONS: -Djava.net.preferIPv4Stack=true
147 sun.tools.jcmd.JCmd
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.)

```
jcmd 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 `jcmd 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 ([source](#)). 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.

```
jcmd <pid> JFR.dump name=continuous_recording filename="/home/recording1.jfr"
```

For more information, please see the [Jcmd Command Reference](#).

## Analyzing Recordings

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](#).

## 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 SSL bindings](#)
- [Add a CDN](#)
- [Configure the Kudu site](#)

### Set Java runtime options

To set allocated memory or other JVM runtime options in both the Tomcat and Java SE environments, create an [app setting](#) named `JAVA_OPTS` with the options. App Service Linux 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` that includes the additional 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`

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.

If you are deploying a JAR application, it should be named `app.jar` so that the built-in image can correctly identify your app. (The Maven plugin does this renaming automatically.) 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. Then paste the full 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`).

### 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>
```

## Adjust startup timeout

If your Java application is particularly large, you should increase the startup time limit. To do this, create an application setting, `WEBSITES_CONTAINER_START_TIME_LIMIT` and set it to the number of seconds that App Service should wait before timing out. The maximum value is `1800` seconds.

## 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 for Linux 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 App Service authentication](#) article.

## 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 users out, use the `/auth/ext/logout` path. To perform other actions, please see the documentation on [App Service Authentication and Authorization usage](#). 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`.

### Spring Boot

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`.

### Configure TLS/SSL

Follow the instructions in the [Secure a custom DNS name with an SSL binding in Azure App Service](#) to upload an existing 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 SSL and TLS.

### 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, please see this documentation on [externalized configurations](#).

### Using 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` .

Additional configuration may be necessary for encrypting your JDBC connection with certificates in the Java Key Store. Please refer to the documentation for your chosen JDBC driver.

- PostgreSQL

- [SQL Server](#)
- [MySQL](#)
- [MongoDB](#)
- [Cassandra](#)

#### Initializing 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, please refer to [the official documentation](#).

## Configure APM platforms

This section shows how to connect Java applications deployed on Azure App Service on Linux with the NewRelic and AppDynamics application performance monitoring (APM) platforms.

#### Configure New Relic

1. Create a NewRelic account at [NewRelic.com](#)
2. Download the Java agent from NewRelic, it will have a file name similar to `newrelic-java-x.x.x.zip`.
3. Copy your license key, you'll need it to configure the agent later.
4. [SSH into your App Service instance](#) and create a new directory `/home/site/wwwroot/apm`.
5. 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`.
6. Modify the YAML file at `/home/site/wwwroot/apm/newrelic/newrelic.yml` and replace the placeholder license value with your own license key.
7. In the Azure portal, browse to your application in App Service and create a new Application Setting.
  - If your app is using **Java SE**, create an environment variable named `JAVA_OPTS` with the value `-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.
  - If you're using **Tomcat**, create an environment variable named `CATALINA_OPTS` with the value `-javaagent:/home/site/wwwroot/apm/newrelic/newrelic.jar`.

#### 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. [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.
  - If you're using **Java SE**, 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.
  - If you're using **Tomcat**, 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 JAR Applications

### Starting JAR Apps

By default, App Service expects your JAR application to be named `app.jar`. If it has this name, it will be run automatically. For Maven users, you can set the JAR name by including `<finalName>app</finalName>` in the `<build>` section of your `pom.xml`. You can do the same in [Gradle](#) by setting the `archiveFileName` property.

If you want to use a different name for your JAR, you must also provide the [Startup Command](#) that executes your JAR file. For example, `java -jar my-jar-app.jar`. You can set the value for your Startup Command in the Portal, under Configuration > General Settings, or with an Application Setting named `STARTUP_COMMAND`.

### Server Port

App Service Linux routes incoming requests to port 80, so your application should listen on port 80 as well. You can do this in your application's configuration (such as Spring's `application.properties` file), or in your Startup Command (for example, `java -jar spring-app.jar --server.port=80`). Please see the following documentation for common Java frameworks:

- [Spring Boot](#)
- [SparkJava](#)
- [Micronaut](#)
- [Play Framework](#)
- [Vertx](#)
- [Quarkus](#)

## Data sources

### 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>

DATABASE	DRIVER CLASS NAME	JDBC DRIVER
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 libssl 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(., '&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>

```

### 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.

## Spring Boot

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_exampledb`.

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_exampledb}
```

Please see the [Spring Boot documentation on data access](#) and [externalized configurations](#) for more information on this topic.

## Use Redis as a session cache with Tomcat

You can configure Tomcat to use an external session store such as [Azure Cache for Redis](#). This enables you to preserve user session state (such as shopping cart data) when a user is transferred to another instance of your app, for example when autoscaling, restart, or failover occurs.

To use Tomcat with Redis, you must configure your app to use a [PersistentManager](#) implementation. The following steps explain this process using [Pivotal Session Manager: redis-store](#) as an example.

1. Open a Bash terminal and use `<variable>=<value>` to set each of the following environment variables.

VARIABLE	VALUE
RESOURCEGROUP_NAME	The name of the resource group containing your App Service instance.
WEBAPP_NAME	The name of your App Service instance.

VARIABLE	VALUE
WEBAPP_PLAN_NAME	The name of your App Service plan.
REGION	The name of the region where your app is hosted.
REDIS_CACHE_NAME	The name of your Azure Cache for Redis instance.
REDIS_PORT	The SSL port that your Redis cache listens on.
REDIS_PASSWORD	The primary access key for your instance.
REDIS_SESSION_KEY_PREFIX	A value that you specify to identify session keys that come from your app.

```
RESOURCEGROUP_NAME=<resource group>
WEBAPP_NAME=<web app>
WEBAPP_PLAN_NAME=<App Service plan>
REGION=<region>
REDIS_CACHE_NAME=<cache>
REDIS_PORT=<port>
REDIS_PASSWORD=<access key>
REDIS_SESSION_KEY_PREFIX=<prefix>
```

You can find the name, port, and access key information on the Azure portal by looking in the **Properties** or **Access keys** sections of your service instance.

2. Create or update your app's *src/main/webapp/META-INF/context.xml* file with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="">
    <!-- Specify Redis Store -->
    <Valve className="com.gopivotal.manager.SessionFlushValve" />
    <Manager className="org.apache.catalina.session.PersistentManager">
        <Store className="com.gopivotal.manager.redis.RedisStore"
            connectionPoolSize="20"
            host="${REDIS_CACHE_NAME}.redis.cache.windows.net"
            port="${REDIS_PORT}"
            password="${REDIS_PASSWORD}"
            sessionKeyPrefix="${REDIS_SESSION_KEY_PREFIX}"
            timeout="2000"
        />
    </Manager>
</Context>
```

This file specifies and configures the session manager implementation for your app. It uses the environment variables that you set in the previous step to keep your account information out of your source files.

3. Use FTP to upload the session manager's JAR file to your App Service instance, placing it in the */home/tomcat/lib* directory. For more info, see [Deploy your app to Azure App Service using FTP/S](#).
4. Disable the **session affinity cookie** for your App Service instance. You can do this from the Azure portal by navigating to your app and then setting **Configuration > General settings > ARR affinity** to **Off**. Alternately, you can use the following command:

```
az webapp update -g <resource group> -n <webapp name> --client-affinity-enabled false
```

By default, App Service will use session affinity cookies to ensure that client requests with existing sessions are routed to the same instance of your application. This default behavior requires no configuration but it can't preserve user session state when your app instance is restarted or when traffic is rerouted to another instance. When you [disable the existing ARR Instance Affinity](#) configuration to turn off the session cookie-based routing, you allow the configured session store to operate without interference.

5. Navigate to the **Properties** section of your App Service instance and find **Additional Outbound IP Addresses**. These represent all possible outbound IP addresses for your app. Copy these for use in the next step.
6. For each IP address, create a firewall rule in your Azure Cache for Redis instance. You can do this on the Azure portal from the **Firewall** section of your Redis instance. Provide a unique name for each rule, and set the **Start IP address** and **End IP address** values to the same IP address.
7. Navigate to the **Advanced settings** section of your Redis instance and set **Allow access only via SSL** to **No**. This enables your App Service instance to communicate with your Redis cache via the Azure infrastructure.
8. Update the `azure-webapp-maven-plugin` configuration in your app's `pom.xml` file to refer to your Redis account info. This file uses the environment variables you set previously to keep your account information out of your source files.

If necessary, change `1.7.0` to the current version of the [Maven Plugin for Azure App Service](#).

```

<plugin>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>azure-webapp-maven-plugin</artifactId>
    <version>1.7.0</version>
    <configuration>

        <!-- Web App information -->
        <resourceGroup>${RESOURCEGROUP_NAME}</resourceGroup>
        <appServicePlanName>${WEBAPP_PLAN_NAME}-${REGION}</appServicePlanName>
        <appName>${WEBAPP_NAME}-${REGION}</appName>
        <region>${REGION}</region>
        <linuxRuntime>tomcat 9.0-jre8</linuxRuntime>

        <appSettings>
            <property>
                <name>REDIS_CACHE_NAME</name>
                <value>${REDIS_CACHE_NAME}</value>
            </property>
            <property>
                <name>REDIS_PORT</name>
                <value>${REDIS_PORT}</value>
            </property>
            <property>
                <name>REDIS_PASSWORD</name>
                <value>${REDIS_PASSWORD}</value>
            </property>
            <property>
                <name>REDIS_SESSION_KEY_PREFIX</name>
                <value>${REDIS_SESSION_KEY_PREFIX}</value>
            </property>
            <property>
                <name>JAVA_OPTS</name>
                <value>-Xms2048m -Xmx2048m -DREDIS_CACHE_NAME=${REDIS_CACHE_NAME} -
DREDIS_PORT=${REDIS_PORT} -DREDIS_PASSWORD=${REDIS_PASSWORD}
IS_SESSION_KEY_PREFIX=${REDIS_SESSION_KEY_PREFIX}</value>
            </property>
        </appSettings>

        </configuration>
    </plugin>

```

## 9. Rebuild and redeploy your app.

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

Your app will now use your Redis cache for session management.

For a sample that you can use to test these instructions, see the [scaling-stateful-java-web-app-on-azure](#) repo on GitHub.

## Docker containers

To use the Azure-supported Zulu JDK in your containers, make sure to pull and use the pre-built images as documented from the [supported Azul Zulu Enterprise for Azure download page](#) or use the [Dockerfile](#) examples from the [Microsoft Java GitHub repo](#).

## Statement of support

### Runtime availability

App Service for Linux supports two runtimes for managed hosting of Java web applications:

- The [Tomcat servlet container](#) for running applications packaged as web archive (WAR) files. Supported versions are 8.5 and 9.0.
- Java SE runtime environment for running applications packaged as Java archive (JAR) files. Supported versions are Java 8 and 11.

### JDK versions and maintenance

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](#).

Supported JDKs are automatically patched on a quarterly basis in January, April, July, and October of each year.

### 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](#).

### 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.

## Next steps

Visit the [Azure for Java Developers](#) center to find Azure quickstarts, tutorials, and Java reference documentation.

General questions about using App Service for Linux that aren't specific to the Java development are answered in the [App Service Linux FAQ](#).

# Configure a Linux Python app for Azure App Service

2/28/2020 • 7 minutes to read • [Edit Online](#)

This article describes how [Azure App Service](#) runs Python apps, 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 processes switched on.

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, you should follow the [Python quickstart](#) and [Python with PostgreSQL tutorial](#) first.

## 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](#).

## Show Python version

To show the current Python 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 Python versions, run the following command in the [Cloud Shell](#):

```
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](#).

## Set Python version

Run the following command in the [Cloud Shell](#) to set the Python version to 3.7:

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --linux-fx-version "PYTHON|3.7"
```

## Customize build automation

If you deploy your app using Git or zip packages with build automation turned on, the App Service build automation steps through the following sequence:

1. Run custom script if specified by `PRE_BUILD_SCRIPT_PATH`.
2. Run `pip install -r requirements.txt`.
3. If `manage.py` is found in the root of the repository, run `manage.py collectstatic`. However, if `DISABLE_COLLECTSTATIC` is set to `true`, this step is skipped.
4. Run custom script if specified by `POST_BUILD_SCRIPT_PATH`.

`PRE_BUILD_COMMAND`, `POST_BUILD_COMMAND`, and `DISABLE_COLLECTSTATIC` 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`. To disable running collectstatic when building Django apps, set `DISABLE_COLLECTSTATIC=true`.

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 Python apps in Linux, see [Oryx documentation: How Python apps are detected and built](#).

## Container characteristics

Python apps deployed to App Service on Linux run within a 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`.
- By default, the base image includes the Flask web framework, but the container supports other frameworks that are WSGI-compliant and compatible with Python 3.7, such as Django.
- To install additional packages, such as Django, create a `requirements.txt` file in the root of your project using `pip freeze > requirements.txt`. Then, publish your project to App Service using Git deployment, which automatically runs `pip install -r requirements.txt` in the container to install your app's dependencies.

## 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 path to 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](#) and replace `<module>` with the name of the module that contains `wsgi.py`.

### Flask app

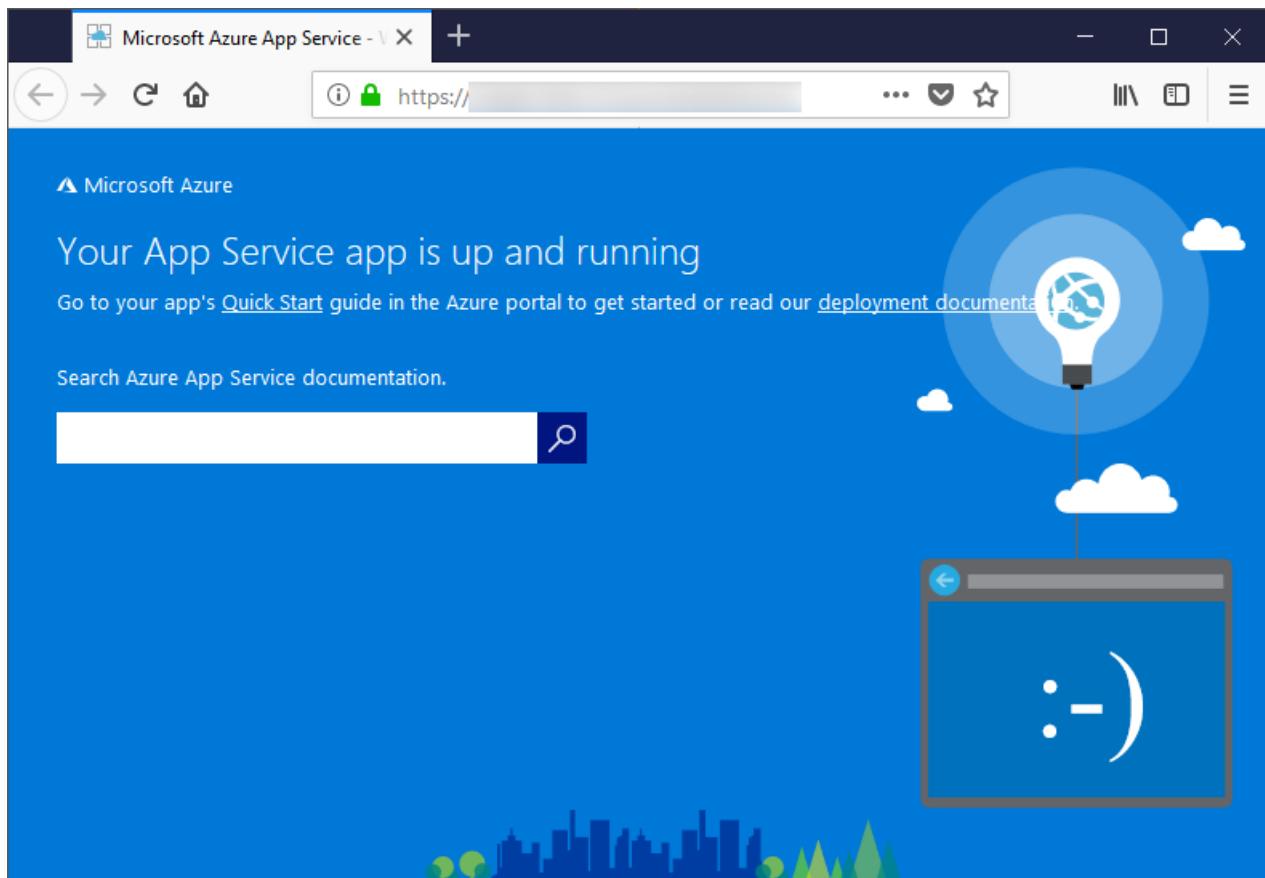
For Flask, App Service looks for a file named `application.py` or `app.py` and starts Gunicorn as follows:

```
# 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. The default app appears as follows:



## Customize startup command

You can control the container's startup behavior by providing a custom Gunicorn startup command. To do this, run the following command in the [Cloud Shell](#):

```
az webapp config set --resource-group <resource-group-name> --name <app-name> --startup-file "<custom-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 *<custom-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
```

You can also add any additional arguments for Gunicorn to `<custom-command>`, such as `--workers=4`. For more information, see [Running Gunicorn](#) (docs.gunicorn.org).

To use a non-Gunicorn server, such as [aiohttp](#), you can replace `<custom-command>` with something like this:

```
python3.7 -m aiohttp.web -H localhost -P 8080 package.module:init_func
```

#### NOTE

App Service ignores any errors that occur when processing a custom command 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 file is deployed to App Service and that it doesn't contain any errors.

## Access environment variables

In App Service, you can [set app settings](#) outside of your app code. Then you can access them using the standard `os.environ` pattern. For example, to access an app setting called `WEBSITE_SITE_NAME`, use the following code:

```
os.environ['WEBSITE_SITE_NAME']
```

## Detect HTTPS session

In App Service, [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 '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 in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

#### 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`.

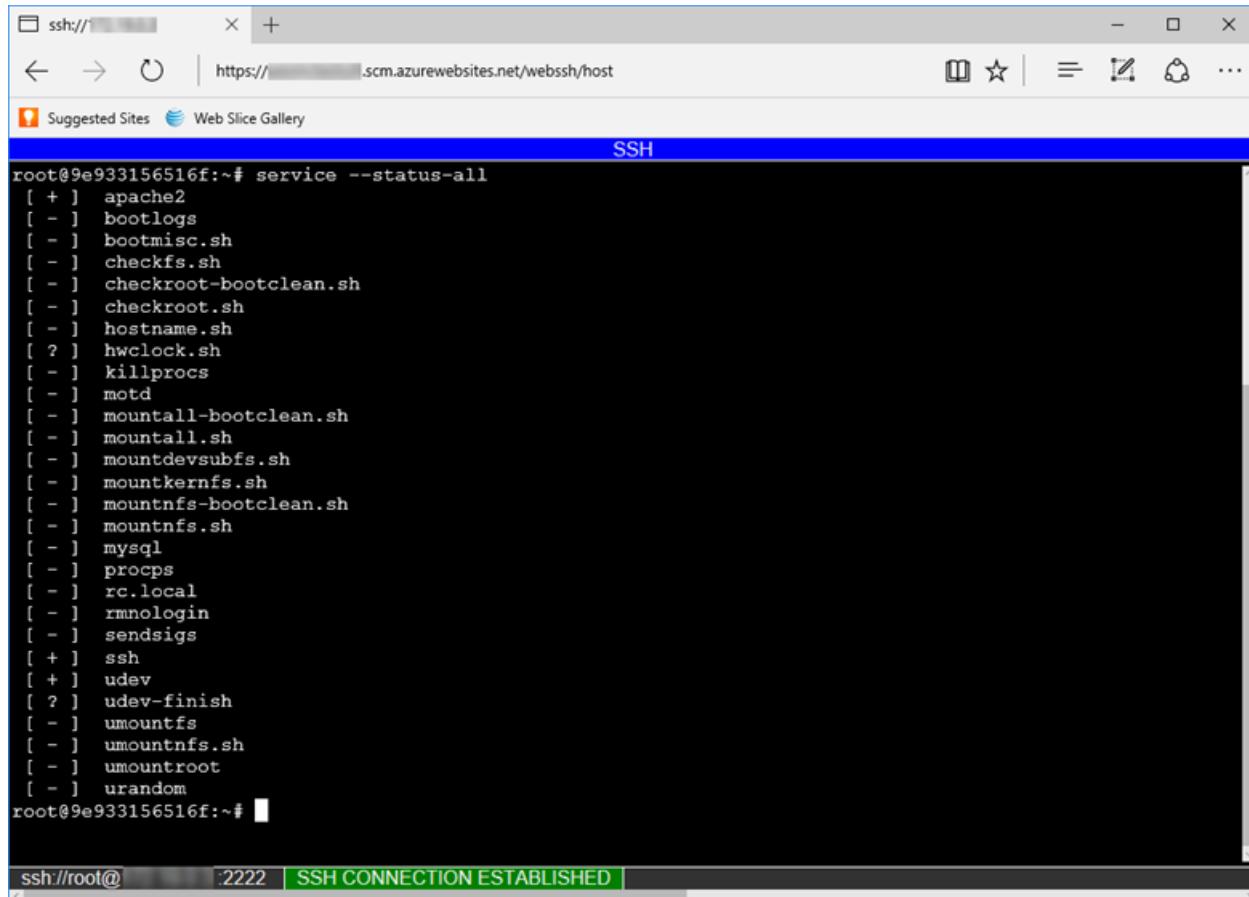
## Open SSH session in browser

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.



The screenshot shows a Microsoft Edge browser window with a tab titled "ssh://". The address bar contains the URL "https://<app-name>.scm.azurewebsites.net/webssh/host". Below the address bar, there are "Suggested Sites" and "Web Slice Gallery" buttons. The main content area is a terminal window titled "SSH". It displays a root shell session with the command "service --status-all" run. The output lists numerous services, mostly marked with a minus sign (-) indicating they are not running. Services listed include apache2, bootlogs, bootmisc.sh, checkfs.sh, checkroot-bootclean.sh, checkroot.sh, hostname.sh, hwclock.sh, killprocs, motd, mountall-bootclean.sh, mountall.sh, mountdebsubfs.sh, mountkernfs.sh, mountnfs-bootclean.sh, mountnfs.sh, mysql, procps, rc.local, rmnologin, sendsigs, ssh, udev, udev-finish, umountfs, umountnfs.sh, umountroot, and urandom. At the bottom of the terminal window, the prompt "root@9e933156516f:~#" is visible. A status bar at the bottom of the browser window indicates "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](#).

## Troubleshooting

- **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_service_name> --query kind`, replacing `<resource_group_name>` and `<app_service_name>` accordingly. You should see `app,linux` as output;

otherwise, recreate the App Service and choose Linux.

- Use SSH or the Kudu console to connect directly to the App Service and verify that your files exist under `/site/wwwroot`. If your files don't exist, review your deployment process 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 arguments that specify the app code are incorrect.
- 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](#).
- [Access the log stream](#).

## Next steps

[Tutorial: Python app with PostgreSQL](#)

[Tutorial: Deploy from private container repository](#)

[App Service Linux FAQ](#)

# Configure a Linux Ruby app for Azure App Service

1/9/2020 • 5 minutes to read • [Edit Online](#)

This article describes how [Azure App Service](#) runs Ruby apps, 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 processes switched on, 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

You can access the console logs generated from inside the container. First, turn on container logging by running the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

### 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`.

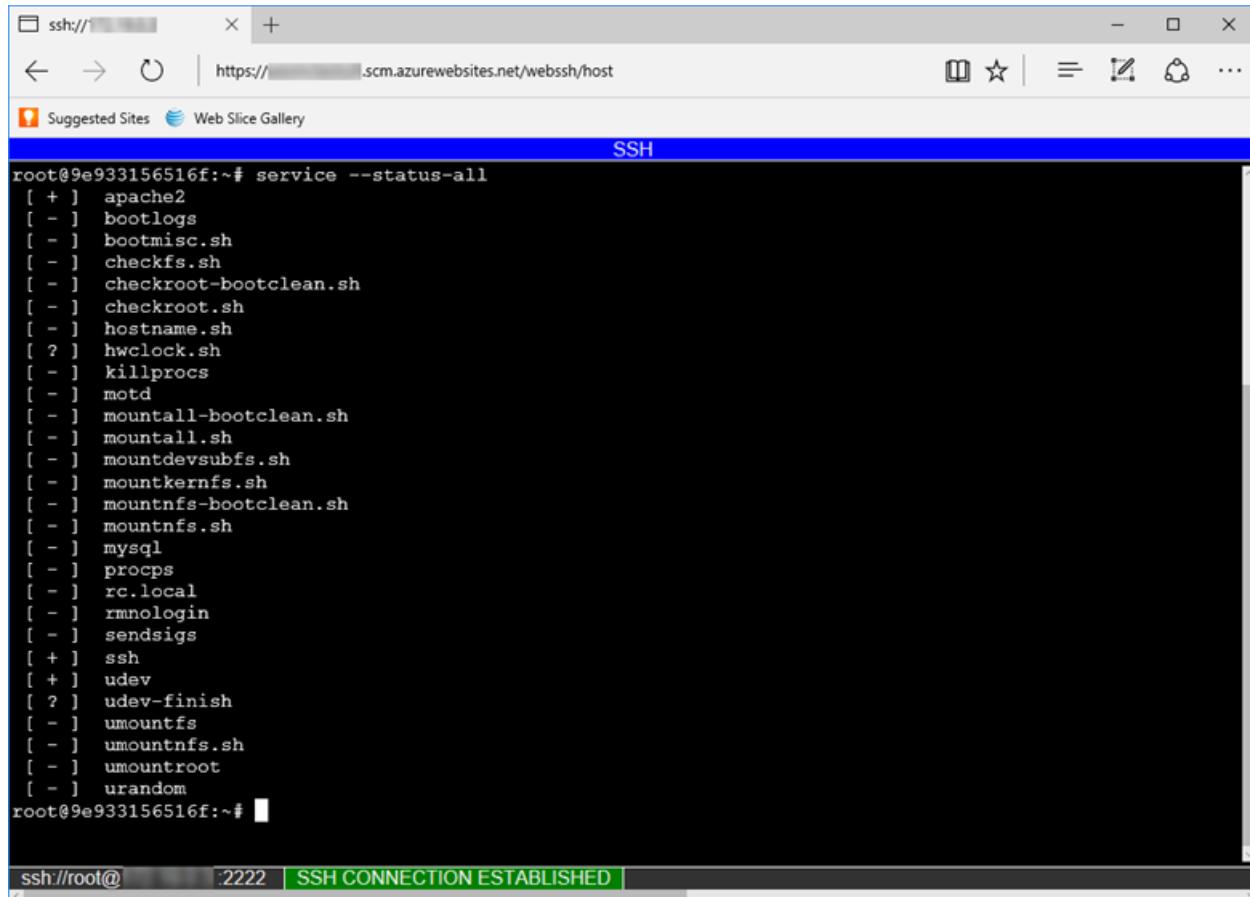
## Open SSH session in browser

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.



The screenshot shows a Microsoft Edge browser window with an in-browser terminal interface. The title bar says "ssh://...". The address bar shows the URL "https://... .scm.azurewebsites.net/webssh/host". Below the address bar are "Suggested Sites" and "Web Slice Gallery" buttons. The main area is titled "SSH" and contains a black terminal window. The terminal output shows the root user listing all services with the command "service --status-all". The output includes many services like apache2, bootlogs, bootmisc.sh, checkfs.sh, checkroot-bootclean.sh, checkroot.sh, hostname.sh, hwclock.sh, killprocs, motd, mountall-bootclean.sh, mountall.sh, mountdebsubfs.sh, mountkernfs.sh, mountnfs-bootclean.sh, mountnfs.sh, mysql, procps, rc.local, rmnologin, sendsigs, ssh, udev, udev-finish, umountfs, umountnfs.sh, umountroot, and urandom. At the bottom of the terminal window, it says "root@9e933156516f:~#". Below the terminal window, a status bar shows "ssh://root@ :2222" and "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](#).

## Next steps

[Tutorial: Rails app with PostgreSQL](#)

[App Service Linux FAQ](#)

# Configure a custom Linux container for Azure App Service

12/2/2019 • 4 minutes to read • [Edit Online](#)

This article shows you how to configure a custom Linux container to run on Azure App Service.

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](#).

## Configure port number

The web server in your custom image may use a port other than 80. You tell Azure about the port that your custom container uses by using the `WEBSITES_PORT` app setting. The GitHub page for the [Python sample in this tutorial](#) shows that you need to set `WEBSITES_PORT` to `8000`. You can set it by running

`az webapp config appsettings set` command in the Cloud Shell. For example:

```
az webapp config appsettings set --resource-group <resource-group-name> --name <app-name> --settings  
WEBSITES_PORT=8000
```

## Configure environment variables

Your custom container may use environment variables that need to be supplied externally. You can pass them in by running `az webapp config appsettings set` command in the Cloud Shell. For example:

```
az webapp config appsettings set --resource-group <resource-group-name> --name <app-name> --settings  
WORDPRESS_DB_HOST="myownserver.mysql.database.azure.com"
```

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 `/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 *enabled* and the setting is not exposed in the Application Settings. To disable it, set the `WEBSITES_ENABLE_APP_SERVICE_STORAGE` app setting by running `az webapp config appsettings set` command in the Cloud Shell. For example:

```
az webapp config appsettings set --resource-group <resource-group-name> --name <app-name> --settings  
WEBSITES_ENABLE_APP_SERVICE_STORAGE=false
```

#### NOTE

You can also [configure your own persistent storage](#).

## 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 the Dockerfile itself.

#### TIP

All built-in Linux containers 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.

- Use the [RUN](#) instruction to install the SSH server and set the password for the root account to "Docker!". For example, for an image based on [Alpine Linux](#), you need the following commands:

```
RUN apk add openssh \
&& echo "root:Docker!" | chpasswd
```

This configuration doesn't allow external connections to the container. SSH is available only through <https://<app-name>.scm.azurewebsites.net> and authenticated with the publishing credentials.

- Add [this sshd\\_config file](#) to your image repository, and use the [COPY](#) instruction to copy the file to the `/etc/ssh/` directory. For more information about `sshd_config` files, see [OpenBSD documentation](#).

```
COPY sshd_config /etc/ssh/
```

#### NOTE

The `sshd_config` file must include the following items:

- `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`.

- Use the [EXPOSE](#) instruction to open port 2222 in the container. Although the root password is known, port 2222 is inaccessible from the internet. It's accessible only by containers within the bridge network of a private virtual network.

```
EXPOSE 80 2222
```

- In the start-up script for your container, start the SSH server.

```
/usr/sbin/sshd
```

For an example, see how the default [Node.js 10.14 container](#) starts the SSH server.

## Access diagnostic logs

You can access the console logs generated from inside the container. First, turn on container logging by running

the following command in the Cloud Shell:

```
az webapp log config --name <app-name> --resource-group myResourceGroup --docker-container-logging filesystem
```

Once container logging is turned on, run the following command to see 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.

#### 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`.

## 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 <resource-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: wordpress:latest  
  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

### 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.

## Configure VNet integration

Using a custom container with VNet integration may require additional container configuration. See [Integrate your app with an Azure Virtual Network](#).

## Next steps

[Tutorial: Deploy from private container repository](#)

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

# Serve content from Azure Storage in App Service on Linux

2/10/2020 • 2 minutes to read • [Edit Online](#)

## NOTE

This article applies to Linux containers. To deploy to custom Windows containers, see [Configure Azure Files in a Windows Container on App Service](#). Azure Storage in App Service on Linux is a **preview** feature. This feature is **not supported for production scenarios**.

This guide shows how to attach Azure Storage to App Service on Linux. Benefits include secured content, content portability, persistent storage, access to multiple apps, and multiple transferring methods.

## Prerequisites

- [Azure CLI](#) (2.0.46 or later).
- An existing [App Service on Linux app](#).
- An [Azure Storage Account](#)
- An [Azure file share and directory](#).

## Limitations of Azure Storage with App Service

- Azure Storage with App Service is **in preview** for App Service on Linux and Web App for Containers. It's **not supported for production scenarios**.
- Azure Storage with App Service supports mounting **Azure Files containers** (Read / Write) and **Azure Blob containers** (Read Only)
- Azure Storage with App Service **doesn't support** using the **Storage Firewall** configuration because of infrastructure limitations.
- Azure Storage with App Service lets you specify **up to five** mount points per app.
- Azure Storage mounted to an app is not accessible through App Service FTP/FTPs endpoints. Use [Azure Storage explorer](#).
- Azure Storage is **not included** with your web app and billed separately. Learn more about [Azure Storage pricing](#).

## WARNING

App Service configurations using Azure Blob Storage will become Read only in Feb 2020. [Learn more](#)

## Configure your app with Azure Storage

Once you've created your [Azure Storage account](#), [file share and directory](#), you can now configure your app with Azure Storage.

To mount a storage account to a directory in your App Service app, you use the `az webapp config storage-account add` command. Storage Type can be AzureBlob or AzureFiles. AzureFiles is used in this example.

Caution

The directory specified as the mount path in your web app should be empty. Any content stored in this directory will be deleted when an external mount is added. If you are migrating files for an existing app, make a backup of your app and its content before you begin.

```
az webapp config storage-account add --resource-group <group_name> --name <app_name> --custom-id <custom_id> -  
-storage-type AzureFiles --share-name <share_name> --account-name <storage_account_name> --access-key "  
<access_key>" --mount-path <mount_path_directory>
```

You should do this for any other directories you want to be linked to a storage account.

## Verify Azure Storage link to the web app

Once a storage container is linked to a web app, you can verify this by running the following command:

```
az webapp config storage-account list --resource-group <resource_group> --name <app_name>
```

## Use Azure Storage in Docker Compose

Azure Storage can be mounted with multi-container apps using the custom-id. To view the custom-id name, run

```
az webapp config storage-account list --name <app_name> --resource-group <resource_group> .
```

In your `docker-compose.yml` file, map the `volumes` option to `custom-id`. For example:

```
wordpress:  
  image: wordpress:latest  
  volumes:  
    - <custom_id>:<path_in_container>
```

## Next steps

- [Configure web apps in Azure App Service.](#)

# Run your app in Azure App Service directly from a ZIP package

2/28/2020 • 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 file

### NOTE

If you downloaded the files in a ZIP file, extract the files first. For example, if you downloaded a ZIP file 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. 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 \(SAS\)](#) 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=bNrVrEFzRHQB17GFJ7boEanetyJ9DGwBSV80M3Mdh%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.

### Use Key Vault References

For added security, you can use Key Vault References in conjunction with your external URL. This keeps the URL encrypted at rest and allows to leverage Key Vault for secret management and rotation. It is recommended to use Azure Blob storage so you can easily rotate the associated SAS key. Azure Blob storage is encrypted at rest, which keeps your application data secure when it is not deployed on App Service.

1. Create an Azure Key Vault.

```
az keyvault create --name "Contoso-Vault" --resource-group <group-name> --location eastus
```

2. Add your external URL as a secret in Key Vault.

```
az keyvault secret set --vault-name "Contoso-Vault" --name "external-url" --value "<insert-your-URL>"
```

3. Create the `WEBSITE_RUN_FROM_PACKAGE` app setting and set 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>")"
```

See the following articles for more information.

- [Key Vault references for App Service](#)
- [Azure Storage encryption for data at rest](#)

## 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.
- 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 your app to Azure App Service using FTP/S

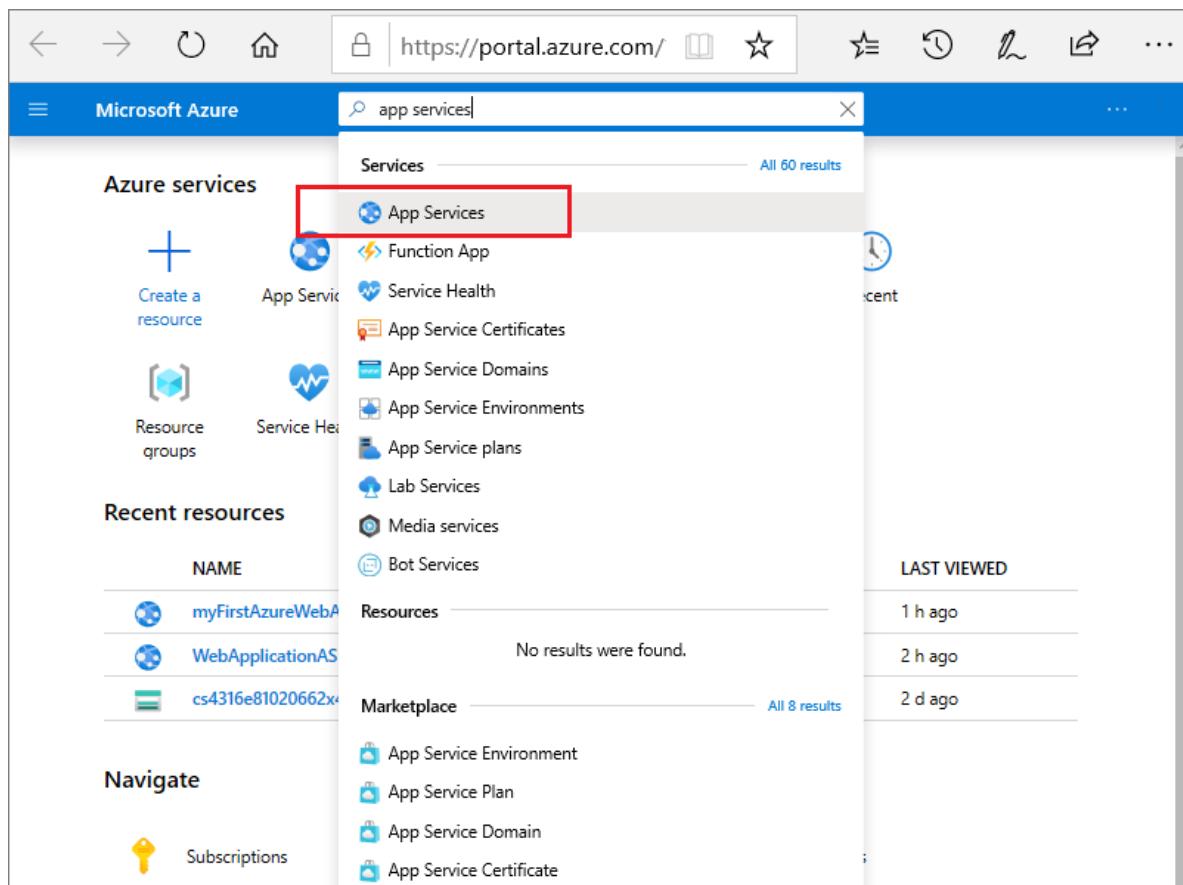
1/6/2020 • 4 minutes to read • [Edit Online](#)

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.

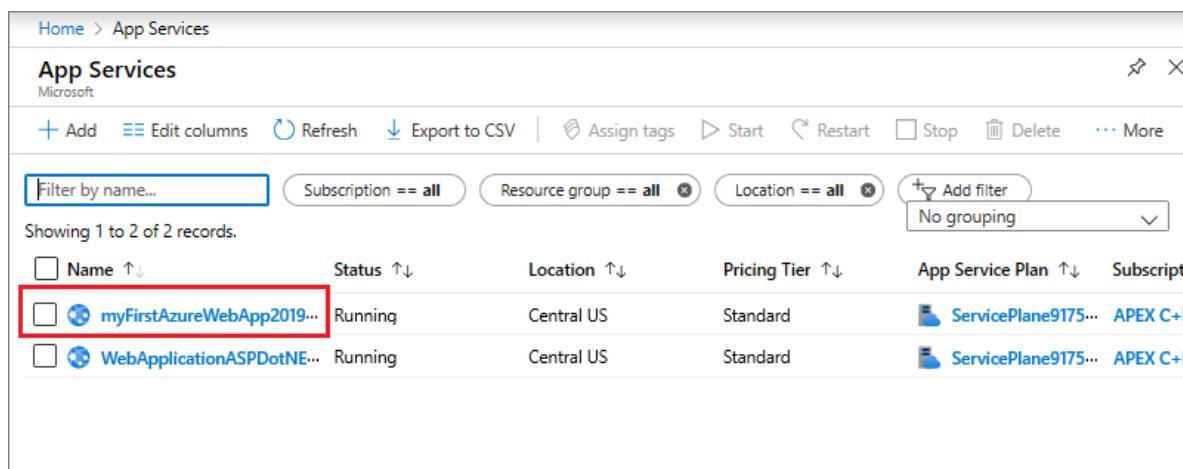
## Open FTP dashboard

1. In the [Azure portal](#), search for and select **App Services**.



The screenshot shows the Microsoft Azure portal interface. The address bar at the top displays the URL <https://portal.azure.com/>. A search bar is present above the main content area, with the text "app services" typed into it. The main content area is titled "Services" and shows a list of items under "All 60 results". The "App Services" item 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", and "Bot Services". To the left of the main content area, there is a sidebar with sections for "Azure services" (Create a resource, Resource groups), "Recent resources" (listing "myFirstAzureWebA", "WebApplicationAS", and "cs4316e81020662x"), and "Navigate" (Subscriptions). On the right side, there is a "LAST VIEWED" section with entries for "1 h ago", "2 h ago", and "2 d ago".

2. Select the web app you want to deploy.



The screenshot shows the "App Services" blade within the Azure portal. The title bar says "Home > App Services". The main area lists two web apps: "myFirstAzureWebApp2019..." and "WebApplicationASPDotNE...". The first app, "myFirstAzureWebApp2019...", is highlighted with a red box. The list includes columns for Name, Status, Location, Pricing Tier, App Service Plan, and Subscription. The "myFirstAzureWebApp2019..." app is listed as "Running" in Central US with a Standard pricing tier and the "ServicePlane9175..." plan. The "WebApplicationASPDotNE..." app is also running in Central US with a Standard pricing tier and the same plan. The top of the blade has various filtering and sorting options, including "Filter by name...", "Subscription == all", "Resource group == all", "Location == all", and "Add filter".

3. Select **Deployment Center > FTP > Dashboard**.

## Get FTP connection information

In the FTP dashboard, select **Copy** to copy the FTPS endpoint and app credentials.

It's recommended that you use **App Credentials** to deploy to your app because it's unique to each app. However, if you click **User Credentials**, you can set user-level credentials that you can use for FTP/S login to all App Service apps in your subscription.

### NOTE

Authenticating to an FTP/FTPS endpoint using user-level credentials requires a username in the following format:

```
<app-name>\<user-name>
```

Since user-level 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.

## 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](#), FTP deployment doesn't support the following deployment automations:

- dependency restores (such as NuGet, NPM, PIP, and Composer automations)
- compilation of .NET binaries
- generation of web.config (here is a [Node.js example](#))

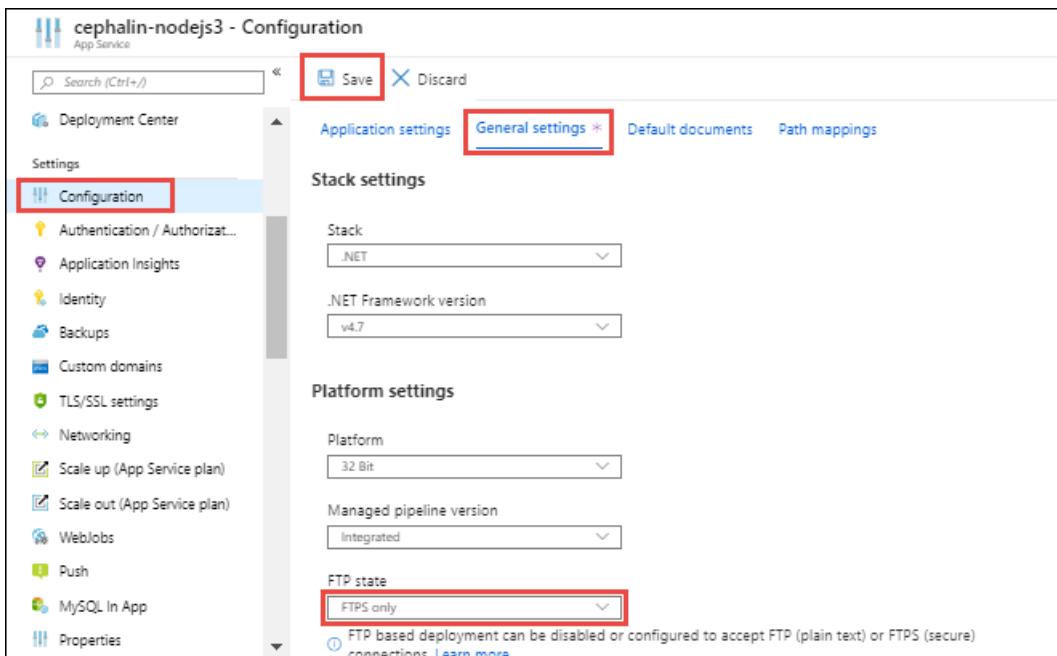
Generate these necessary files manually on your local machine, and then deploy them together with your app.

## Enforce FTPS

For enhanced security, you should allow FTP over SSL only. You can also disable both FTP and FTPS if you don't use FTP deployment.

In your app's resource page in [Azure portal](#), select **Configuration > General settings** from the left navigation.

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**.



## Automate with scripts

For FTP deployment using [Azure CLI](#), see [Create a web app and deploy files with FTP \(Azure CLI\)](#).

For FTP deployment using [Azure PowerShell](#), see [Upload files to a web app using FTP \(PowerShell\)](#).

## 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. Therefore, the deployment can fail because of locked files. The app

may also behave unpredictably during deployment, because not all the files updated at the same time. This is undesirable for a customer-facing app. There are a few different ways to avoid these issues:

- [Run your app from the ZIP package directly](#) without unpacking it.
- Stop your app or enable offline mode for your app during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) enabled.

## 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
- 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](#).

## 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

- [Azure App Service Deployment Credentials](#)

# Sync content from a cloud folder to Azure App Service

2/20/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to sync your content to [Azure App Service](#) from Dropbox and OneDrive.

The on-demand content sync deployment is powered by the App Service [Kudu deployment engine](#). You can work with your app code and content in a designated cloud folder, and then sync to App Service with the click of a button. Content sync uses the Kudu build server.

## Enable content sync deployment

To enable content sync, navigate to your App Service app page in the [Azure portal](#).

In the left menu, click **Deployment Center** > **OneDrive** or **Dropbox** > **Authorize**. Follow the authorization prompts.

The screenshot shows the Azure Deployment Center interface for the app 'deploymentcenter-demo'. The 'Deployment Center' tab is selected in the left sidebar. In the main area, there are several deployment options:

- Bitbucket**: Configure continuous integration with a Bitbucket repo. Status: Not Authorized.
- Local Git**: Deploy from a local Git repo.
- OneDrive**: Sync content from a OneDrive cloud folder. Status: Not Authorized. This option is highlighted with a red box.
- Dropbox**: Sync content from a Dropbox cloud folder. Status: Not Authorized. This option is highlighted with a red box.
- External**: Deploy from a public Git or Mercurial repo.
- FTP**: Use an FTP connection to access and copy app files.

A blue 'Authorize' button is located at the bottom right of the main content area, also highlighted with a red box.

You only need to authorize with OneDrive or Dropbox once. If you're already authorized, just click **Continue**. You can change the authorized OneDrive or Dropbox account by clicking **Change account**.

The screenshot shows the Azure Deployment Center interface. On the left, there's a sidebar with links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (with Quickstart, Deployment slots (Preview), Deployment slots, Deployment options (Classic), and Deployment Center selected), Application settings, Authentication / Authorization, Application Insights, Managed service identity, Backups, Custom domains, and SSL settings. The main area has a search bar at the top. It lists several deployment methods: Bitbucket (disabled), Local Git (disabled), OneDrive (selected and highlighted with a red box), Dropbox (selected and highlighted with a red box), External (disabled), and FTP (disabled). At the bottom are 'Change Account' and 'Continue' buttons, with 'Continue' also highlighted with a red box.

In the **Configure** page, select the folder you want to synchronize. This folder is created under the following designated content path in OneDrive or Dropbox.

- **OneDrive:** Apps\Azure Web Apps
- **Dropbox:** Apps\Azure

When finished, click **Continue**.

In the **Summary** page, verify your options and click **Finish**.

## Synchronize content

When you want to synchronize content in your cloud folder with App Service, go back to the **Deployment Center** page and click **Sync**.

Dashboard > deploymentcenter-demo - Deployment Center

**deploymentcenter-demo - Deployment Center**

App Service

Search (Ctrl + /) Refresh Disconnect Sync Deployment Credentials

Overview Activity log Access control (IAM) Tags Diagnose and solve problem...

Deployment Quickstart Deployment slots (Preview) Deployment slots Deployment options (Classic) Deployment Center

Source Dropbox Folder /Apps/Azure/deploymentcenter-demo

Build Kudu Rollback Enabled No

TIME	STATUS	COMMIT ID (AUTI)	CHECKIN MESSAGE	LOGS
Monday, December 3, 2018	Success (Active)	8d9ee47 (Cephas L)	Synchronized 0 change(s) from Dropbox.	

Settings Application settings Authentication / Authorizati... Application Insights Managed service identity Backups Custom domains SSL settings

The screenshot shows the Azure Deployment Center for the app 'deploymentcenter-demo'. The 'Sync' button in the top navigation bar is highlighted with a red box. The main area displays deployment details: Source is Dropbox, Build is Kudu, and the commit message is 'Synchronized 0 change(s) from Dropbox.' A note at the bottom states: 'Because of underlying differences in the APIs, OneDrive for Business is not supported at this time.'

#### NOTE

Because of underlying differences in the APIs, **OneDrive for Business** is not supported at this time.

## Disable content sync deployment

To disable content sync, navigate to your App Service app page in the [Azure portal](#).

In the left menu, click **Deployment Center > Disconnect**.

The screenshot shows the Azure Deployment Center for the 'deploymentcenter-demo' app service. The left sidebar lists navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Deployment (Quickstart, Deployment slots), Deployment options (Classic), and Deployment Center (which is selected and highlighted with a red box). The main content area displays deployment details: Source is Dropbox, Build is Kudu, and the folder is /Apps/Azure/deploymentcenter-demo. It shows a build history table with one entry: TIME (Monday, December 3, 2018), STATUS (Success (Active)), COMMIT ID (8d9ee47 (Cephas L)), and CHECKIN MESSAGE (Synchronized 0 change(s) from Dropbox). A red box highlights the 'Disconnect' button in the top right toolbar.

## 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. Therefore, the deployment can fail because of locked files. The app may also behave unpredictably during deployment, because not all the files updated at the same time. This is undesirable for a customer-facing app. There are a few different ways to avoid these issues:

- Run your app from the ZIP package directly without unpacking it.
- Stop your app or enable offline mode for your app during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a [staging slot](#) with [auto swap](#) enabled.

## Next steps

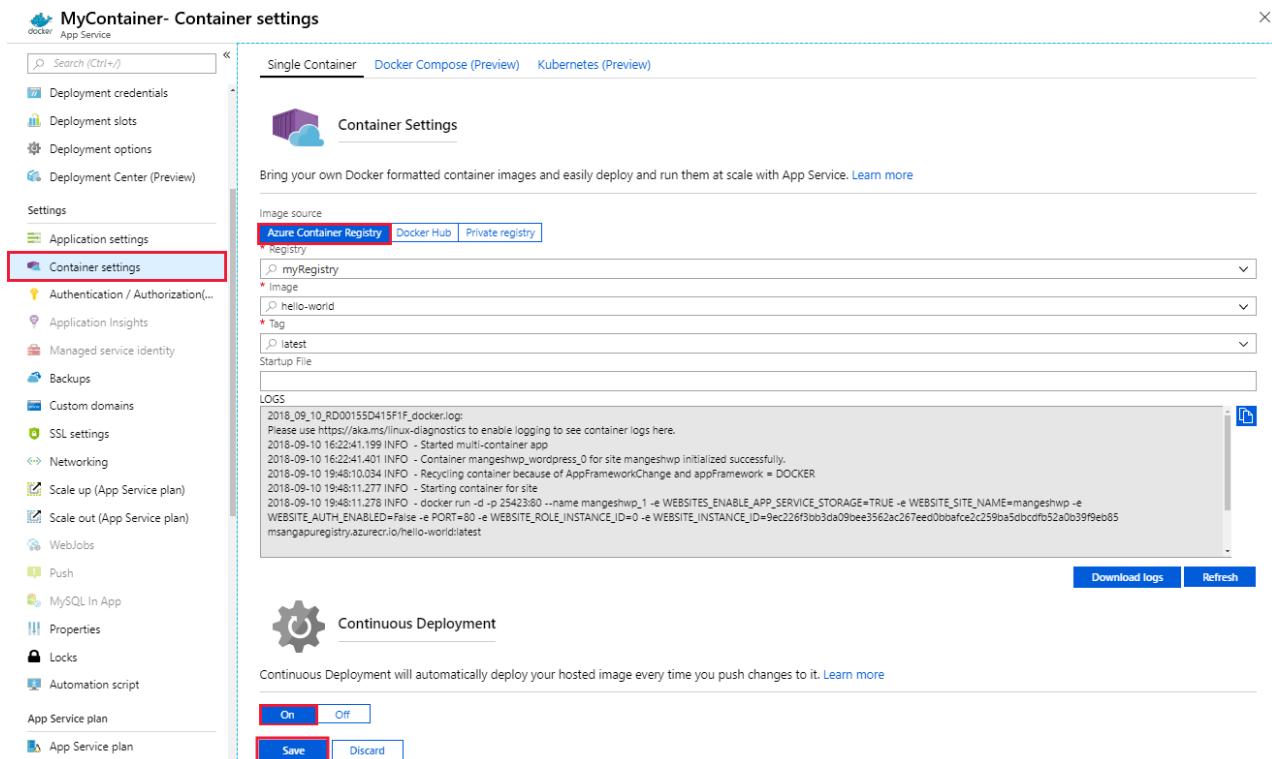
[Deploy from local Git repo](#)

# Continuous deployment with Web App for Containers

12/2/2019 • 2 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](#).

## Enable continuous deployment with ACR



The screenshot shows the Azure Portal interface for configuring continuous deployment. On the left, a sidebar lists various app service settings like Application settings, Container settings, and Container logs. The 'Container settings' section is currently selected. The main pane displays the 'Container Settings' configuration, specifically for using an Azure Container Registry. It shows the registry 'myRegistry' and the image 'hello-world' with tag 'latest'. Below this, the 'LOGS' section displays deployment logs for a Docker container. At the bottom, the 'Continuous Deployment' section has the 'On' toggle switch selected and the 'Save' button highlighted.

1. Sign in to the [Azure portal](#).
2. Select the **App Service** option on the left side of the page.
3. Select the name of the app for which you want to configure continuous deployment.
4. On the **Container Settings** page, select **Single Container**
5. Select **Azure Container Registry**
6. Select **Continuous Deployment > On**
7. Select **Save** to enable continuous deployment.

## Use the ACR webhook

Once Continuous Deployment has been enabled, you can view the newly created webhook on your Azure Container Registry webhooks page.

The screenshot shows the Azure Container Registry interface for a registry named 'myRegistry'. On the left, there's a sidebar with various navigation links: Overview, Activity log, Access control (IAM), Tags, Quick start, Events, Settings (with sub-links for Access keys, Locks, and Automation script), Services (with sub-links for Repositories and Webhooks), and Replications. The 'Webhooks' link is highlighted with a red box. The main content area has a search bar at the top and a table below it. The table columns are NAME, LOCATION, ACTIONS, SCOPE, STATUS, and ... (ellipsis). One row is visible: 'automaticwebho...' under NAME, 'South Central US' under LOCATION, 'push' under ACTIONS, 'foo' under SCOPE, 'On' under STATUS, and a green checkmark icon under the ellipsis.

In your Container Registry, click on Webhooks to view the current webhooks.

## Enable continuous deployment with Docker Hub (optional)

1. Sign in to the [Azure portal](#).
2. Select the **App Service** option on the left side of the page.
3. Select the name of the app for which you want to configure continuous deployment.
4. On the **Container Settings** page, select **Single Container**
5. Select **Docker Hub**
6. Select **Continuous Deployment > On**
7. Select **Save** to enable continuous deployment.

Home > MyContainer - Container settings

## MyContainer- Container settings

Search (Ctrl+)

Overview Activity log Access control (IAM) Tags Diagnose and solve problems

Deployment

- Quickstart
- Deployment credentials
- Deployment slots
- Deployment options
- Deployment Center (Preview)

Settings

- Application settings
- Container settings**
- Authentication / Authorization(...)
- Application Insights
- Managed service identity
- Backups
- Custom domains
- SSL settings
- Networking
- Scale up (App Service plan)
- Scale out (App Service plan)
- WebJobs
- Push

Bring your own Docker formatted container images and easily deploy and run them at scale with App Service. [Learn more](#)

Image source [Azure Container Registry](#) [Docker Hub](#) [Private registry](#)

Repository Access [Public](#) [Private](#)

\* Image and optional tag (eg 'imagetag')

Startup File

LOGS

```
2018_09_07_RD2818780E252E_docker.log:  
Digest: sha256:ea6844f81b57ccce7314a1f7e774bd8a30b4baca2330704fc50119319147c4  
Status: Downloaded newer image for redis:3-alpine  
  
2018-09-07 17:33:09.266 INFO - Starting container for site  
2018-09-07 17:33:09.367 INFO - docker run -d -p 0:6379 --name mangeshwp_redis_0 -e WEBSITES_ENABLE_APP_SERVICE_STORAGE=TRUE -e WEBSITE_SITE_NAME=mangeshwp -e WEBSITE_AUTH_ENABLED=False -e WEBSITE_ROLE_INSTANCE_ID=0 -e WEBSITE_INSTANCE_ID=7d642622a36794490195a26f56722396a95025f63c206c59b65cf93c634da18d redis:3-alpine  
  
2018-09-07 17:33:09.368 INFO - Logging is not enabled for this container.  
Please use https://aka.ms/linux-diagnostics to enable logging to see container logs here.
```

[Download logs](#) [Refresh](#)

### Continuous Deployment

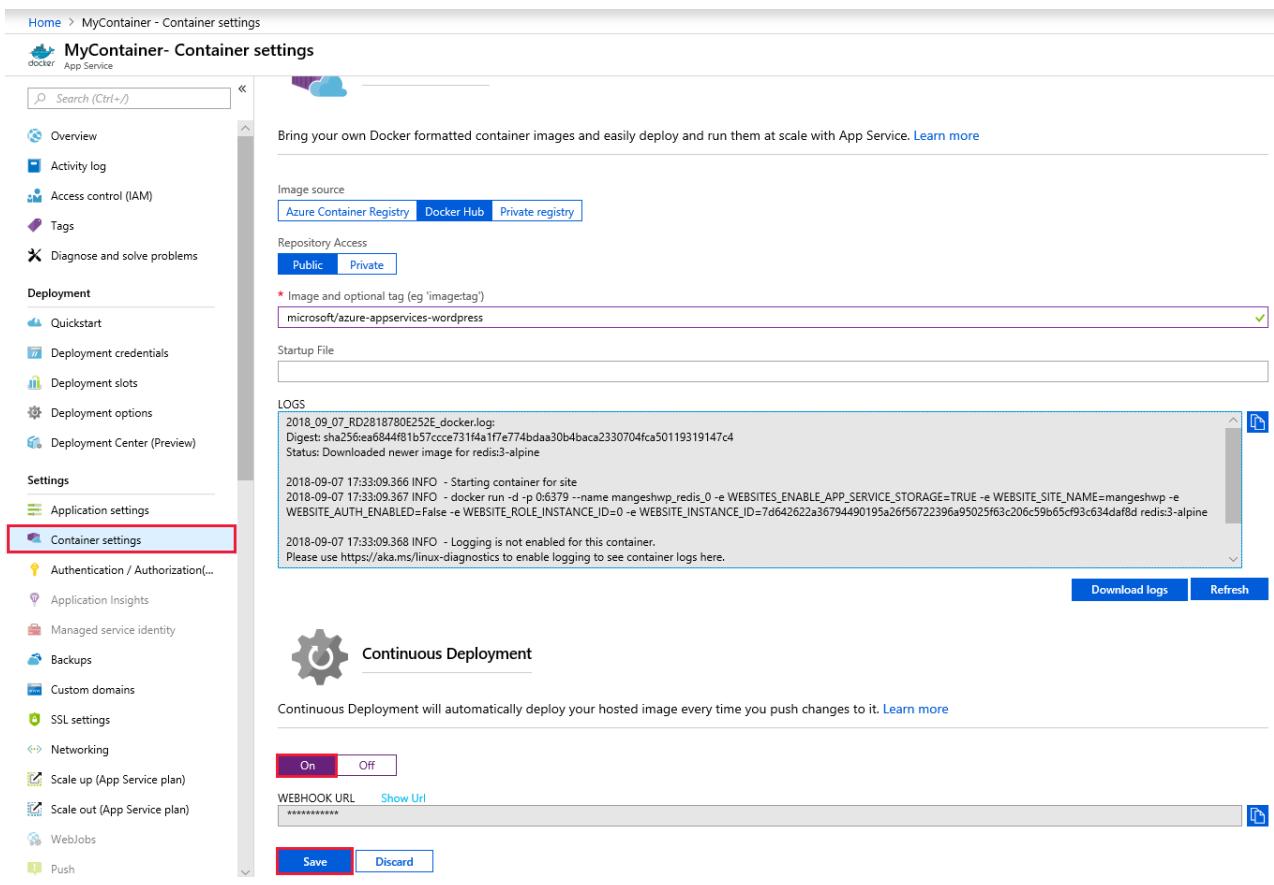
Continuous Deployment will automatically deploy your hosted image every time you push changes to it. [Learn more](#)

On Off

WEBHOOK URL [Show Url](#)

\*\*\*\*\*

[Save](#) [Discard](#)



Copy the Webhook URL. To add a webhook for Docker Hub, follow [webhooks for Docker Hub](#).

## Next steps

- [Introduction to Azure App Service on Linux](#)
- [Azure Container Registry](#)
- [Create a .NET Core web app in App Service on Linux](#)
- [Create a Ruby web app in App Service on Linux](#)
- [Deploy a Docker/Go web app in Web App for Containers](#)
- [Azure App Service on Linux FAQ](#)
- [Manage Web App for Containers using Azure CLI](#)

# Local Git deployment to Azure App Service

2/11/2020 • 8 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 automatic builds from Azure App Service Kudu 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](#).

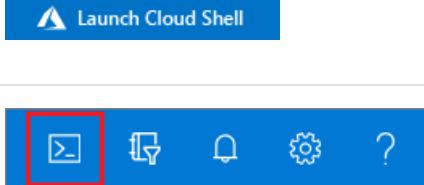
#### NOTE

If you develop in Visual Studio, let [Visual Studio create a repository for you](#). The project is immediately ready to be deployed by using Git.

## 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 <b>Try It</b> in the upper-right corner of a code block. Selecting <b>Try It</b> 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 <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> 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 with Kudu build server

The easiest way to enable local Git deployment for your app with the Kudu App Service build server is to use Azure Cloud Shell.

### 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.

## Get the deployment URL

To get the URL to enable local Git deployment for an existing app, run

```
az webapp deployment source config-local-git
```

 in the Cloud Shell. Replace `<app-name>` and `<group-name>` with the names of your app and its Azure resource group.

```
az webapp deployment source config-local-git --name <app-name> --resource-group <group-name>
```

### NOTE

If you are using a linux app-service-plan, you need to add this parameter: `--runtime python|3.7`

Or, to create a new Git-enabled app, run `az webapp create` in the Cloud Shell with the `--deployment-local-git` parameter. Replace `<app-name>`, `<group-name>`, and `<plan-name>` with the names for your new Git app, its Azure resource group, and its Azure App Service plan.

```
az webapp create --name <app-name> --resource-group <group-name> --plan <plan-name> --deployment-local-git
```

Either command returns 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.

Instead of using this account-level URL, you can also enable local Git by using app-level credentials. Azure App Service automatically generates these credentials for every app.

Get the app credentials by running the following command in the Cloud Shell. Replace `<app-name>` and `<group-name>` with your app's name and Azure resource group name.

```
az webapp deployment list-publishing-credentials --name <app-name> --resource-group <group-name> --query scmUri --output tsv
```

Use the URL that returns to deploy your app in the next step.

## Deploy the web app

1. Open a local terminal window to your local Git repository, and add an Azure remote. In the following command, replace `<url>` with the deployment user-specific URL or app-specific URL you got from the previous step.

```
git remote add azure <url>
```

2. Push to the Azure remote with `git push azure master`.
3. In the **Git Credential Manager** window, enter your **deployment user password**, not your Azure sign-in password.

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.

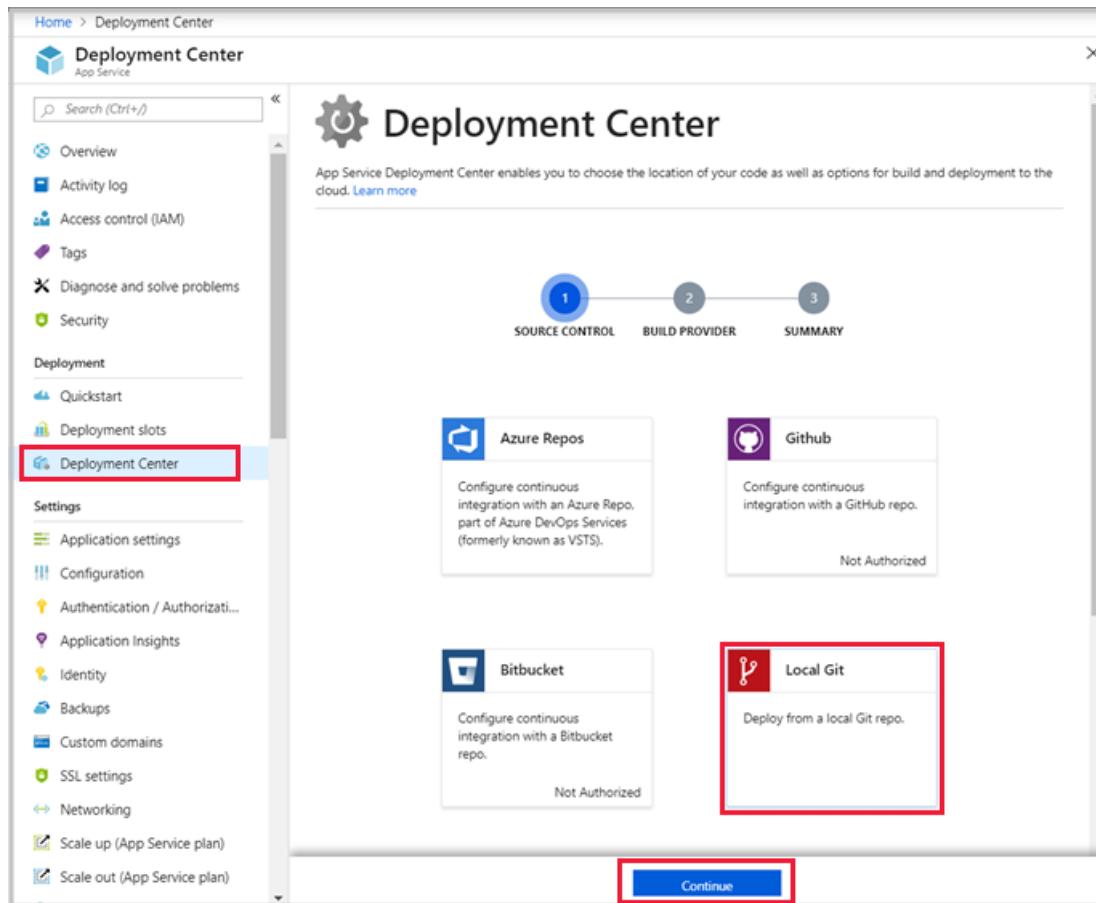
## Deploy with Azure Pipelines builds

If your account has the necessary permissions, you can set up Azure Pipelines (Preview) to enable local Git deployment for your app.

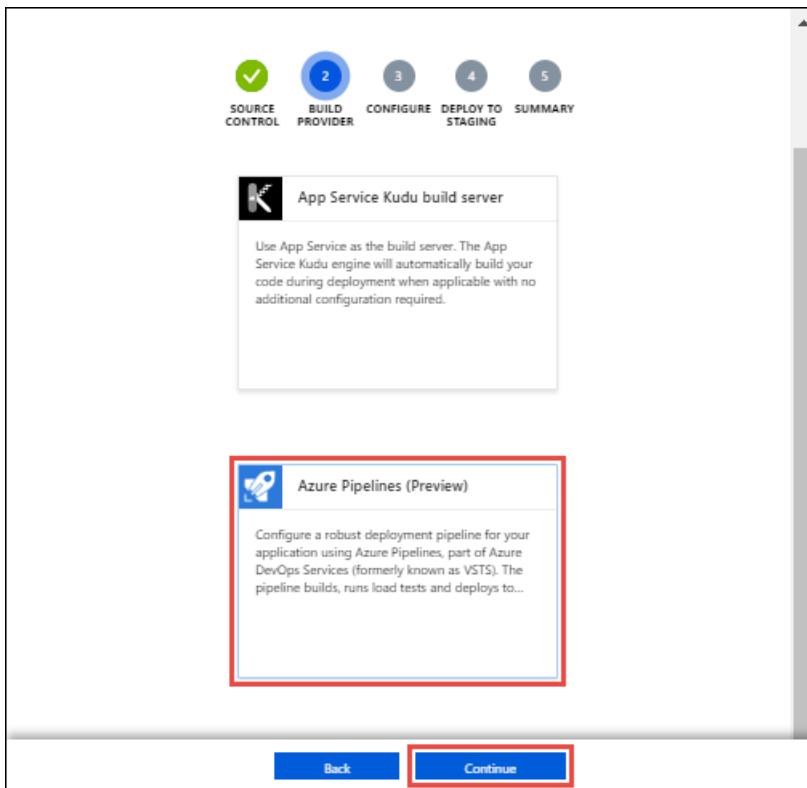
- Your Azure account must have permissions to write to Azure Active Directory and create a service.
- Your Azure account must have the **Owner** role in your Azure subscription.
- You must be an administrator in the Azure DevOps project you want to use.

To enable local Git deployment for your app with Azure Pipelines (Preview):

1. In the [Azure portal](#), search for and select **App Services**.
2. Select your Azure App Service app and select **Deployment Center** in the left menu.
3. On the **Deployment Center** page, select **Local Git**, and then select **Continue**.



4. On the **Build provider** page, select **Azure Pipelines (Preview)**, and then select **Continue**.



5. On the **Configure** page, configure a new Azure DevOps organization, or specify an existing organization, and then select **Continue**.

**NOTE**

If your existing Azure DevOps organization isn't listed, you may need to link it to your Azure subscription. For more information, see [Define your CD release pipeline](#).

6. Depending on your App Service plan [pricing tier](#), you may see a **Deploy to staging** page. Choose whether to [enable deployment slots](#), and then select **Continue**.
7. On the **Summary** page, review the settings, and then select **Finish**.
8. When the Azure Pipeline is ready, copy the Git repository URL from the **Deployment Center** page to use in the next step.

The screenshot shows the Azure Deployment Center interface. On the left, there's a sidebar with navigation links like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots), and Deployment Center (Settings: Application settings, Configuration, Authentication / Authorizati..., Application Insights, Identity, Backups, Custom domains, SSL settings). The main area displays repository details: Build (Azure Pipelines), Account (demo), Source (Azure Repos (TfsGit)), Project (deploymentcenter-demo), Repository URL ([https://deployment-center-demo@dev.azure.com/demo/demo/\\_git/demo](https://deployment-center-demo@dev.azure.com/demo/demo/_git/demo)), Branch (master), Slot (Production). Below this, a log entry for Monday, March 4, 2019, shows a successful setup of Continuous Delivery for the repository at 4:29:53 PM GMT-6.

9. In your local terminal window, add an Azure remote to your local Git repository. In the command, replace <url> with the URL of the Git repository that you got from the previous step.

```
git remote add azure <url>
```

10. Push to the Azure remote with `git push azure master`.
11. On the **Git Credential Manager** page, sign in with your visualstudio.com username. For other authentication methods, see [Azure DevOps Services authentication overview](#).
12. Once deployment is finished, view the build progress at [https://<azure\\_devops\\_account>.visualstudio.com/<project\\_name>/\\_build](https://<azure_devops_account>.visualstudio.com/<project_name>/_build), and the deployment progress at [https://<azure\\_devops\\_account>.visualstudio.com/<project\\_name>/\\_release](https://<azure_devops_account>.visualstudio.com/<project_name>/_release).
13. Browse to your app in the Azure portal to verify that the content is deployed.

## 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. Therefore, the deployment can fail because of locked files. The app may also behave unpredictably during deployment, because not all the files updated at the same time. This is undesirable for a customer-facing app. There are a few different ways to avoid these issues:

- Run your app from the ZIP package directly without unpacking it.
- Stop your app or enable offline mode for your app during deployment. For more information, see [Deal with locked files during deployment](#).
- Deploy to a staging slot with [auto swap](#) enabled.

## 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
Unable to access '[siteURL]': Failed to connect to [scmAddress]	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.
Couldn't resolve host 'hostname'	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.
No refs in common and none specified; doing nothing. Perhaps you should specify a branch such as 'master'.	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 master branch: <code>git push azure master</code> .
src refspec [branchname] does not match any.	You tried to push to a branch other than master on the 'azure' remote.	Run <code>git push</code> again, specifying the master branch: <code>git push azure master</code> .
RPC failed; result=22, HTTP code = 5xx.	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.	<p>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:</p> <p><b>Malformed package.json file:</b>  <code>npm ERR! Couldn't read dependencies.</code></p> <p><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></p>

## Additional resources

- [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 a custom container to App Service using GitHub Actions

2/21/2020 • 3 minutes to read • [Edit Online](#)

[GitHub Actions](#) gives you the flexibility to build an automated software development lifecycle workflow. With the [Azure App Service Action for Containers](#), you can automate your workflow to deploy apps as [custom containers to App Service](#) using GitHub Actions.

## IMPORTANT

GitHub Actions is currently in beta. You must first [sign-up to join the preview](#) using your GitHub account.

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.

For an Azure App Service container workflow, the file has three sections:

SECTION	TASKS
<b>Authentication</b>	1. Define a service principal. 2. Create a GitHub secret.
<b>Build</b>	1. Set up the environment. 2. Build the container image.
<b>Deploy</b>	1. Deploy the container image.

## Create a service principal

You can create a [service principal](#) by using the `az ad sp create-for-rbac` command in the [Azure CLI](#). You can run this command using [Azure Cloud Shell](#) in the Azure portal or by selecting the **Try it** button.

```
az ad sp create-for-rbac --name "myApp" --role contributor \
    --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group} \
    --sdk-auth

# Replace {subscription-id}, {resource-group} with the subscription, resource group details of the WebApp
```

The output is a JSON object with the role assignment credentials that provide access to your App Service app similar to below. Copy this JSON object to authenticate from GitHub.

```
{
  "clientId": "<GUID>",
  "clientSecret": "<GUID>",
  "subscriptionId": "<GUID>",
  "tenantId": "<GUID>",
  (...)
```

## IMPORTANT

It is always a good practice to grant minimum access. You could restrict scope in the above Az CLI command to the specific App Service app and the Azure Container Registry where the container images are pushed to.

## Configure the GitHub secret

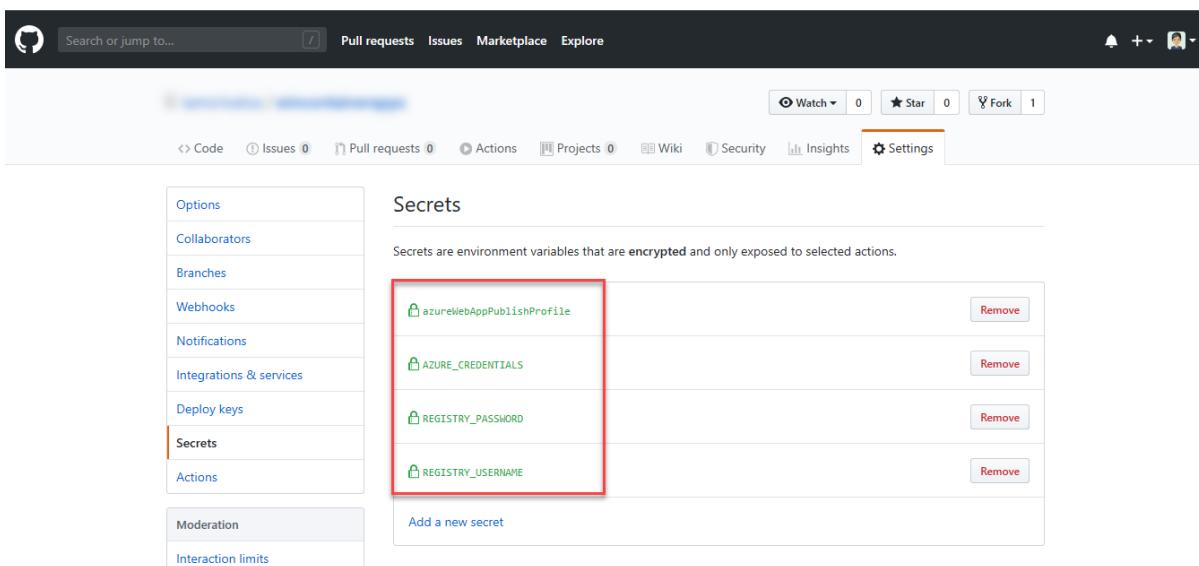
The below example uses user-level credentials i.e. Azure Service Principal for deployment. Follow the steps to configure the secret:

1. In [GitHub](#), browse your repository, select **Settings > Secrets > Add a new secret**
2. Paste the contents of the below `az cli` command as the value of secret variable. For example, `AZURE_CREDENTIALS`.

```
az ad sp create-for-rbac --name "myApp" --role contributor \
    --scopes /subscriptions/{subscription-id}/resourceGroups/{resource-group} \
    --sdk-auth

# Replace {subscription-id}, {resource-group} with the subscription, resource group details
```

3. Now in the workflow file in your branch: `.github/workflows/workflow.yml` replace the secret in Azure login action with your secret.
4. Similarly, define the following additional secrets for the container registry credentials and set them in Docker login action.
  - `REGISTRY_USERNAME`
  - `REGISTRY_PASSWORD`
5. You see the secrets as shown below once defined.



## Build the Container image

The following example show part of the workflow that builds the docker image.

```

on: [push]

name: Linux_Container_Node_Workflow

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      # checkout the repo
      - name: 'Checkout GitHub Action'
        uses: actions/checkout@master

      - name: 'Login via Azure CLI'
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - uses: azure/docker-login@v1
        with:
          login-server: contoso.azurecr.io
          username: ${{ secrets.REGISTRY_USERNAME }}
          password: ${{ secrets.REGISTRY_PASSWORD }}

      - run: |
        docker build . -t contoso.azurecr.io/nodejssampleapp:${{ github.sha }}
        docker push contoso.azurecr.io/nodejssampleapp:${{ github.sha }}

```

## Deploy to an App Service container

To deploy your image to a custom container in App Service, use the `azure/webapps-container-deploy@v1` action. This action has five parameters:

PARAMETER	EXPLANATION
<b>app-name</b>	(Required) Name of the App Service app
<b>slot-name</b>	(Optional) Enter an existing Slot other than the Production slot
<b>images</b>	(Required) 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) 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>container-command</b>	(Optional) Enter the start-up command. For ex. dotnet run or dotnet filename.dll

Below is the sample workflow to build and deploy a Node.js app to a custom container in App Service.

```

on: [push]

name: Linux Container Node Workflow

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
      # checkout the repo
      - name: 'Checkout GitHub Action'
        uses: actions/checkout@master

      - name: 'Login via Azure CLI'
        uses: azure/login@v1
        with:
          creds: ${{ secrets.AZURE_CREDENTIALS }}

      - uses: azure/docker-login@v1
        with:
          login-server: contoso.azurecr.io
          username: ${{ secrets.REGISTRY_USERNAME }}
          password: ${{ secrets.REGISTRY_PASSWORD }}

      - run: |
          docker build . -t contoso.azurecr.io/nodejssampleapp:${{ github.sha }}
          docker push contoso.azurecr.io/nodejssampleapp:${{ github.sha }}

      - uses: azure/webapps-container-deploy@v1
        with:
          app-name: 'node-rnc'
          images: 'contoso.azurecr.io/nodejssampleapp:${{ github.sha }}'

      - name: Azure logout
        run: |
          az logout

```

## 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.

- [Azure login](#)
- [Azure WebApp](#)
- [Azure WebApp for containers](#)
- [Docker login/logout](#)
- [Events that trigger workflows](#)
- [K8s deploy](#)
- [Starter CI Workflows](#)
- [Starter workflows to deploy to Azure](#)

# Provision and deploy microservices predictably in Azure

1/14/2020 • 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 templatize 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

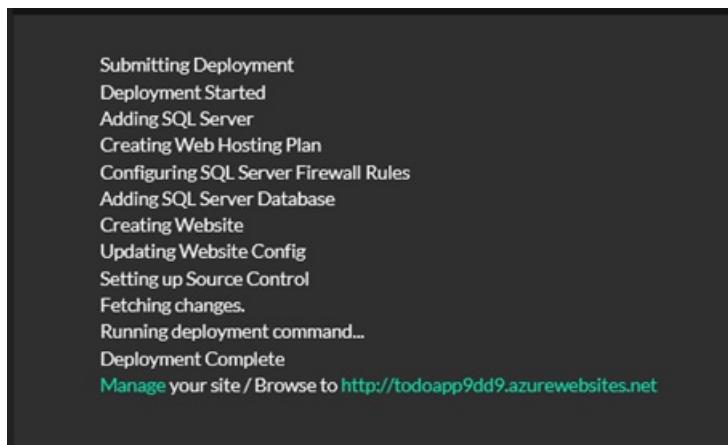
So now let's get right to it.

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**.

Git Repository Url - <https://github.com/azure-appservice-samples/ToDoApp>  
Branch - master

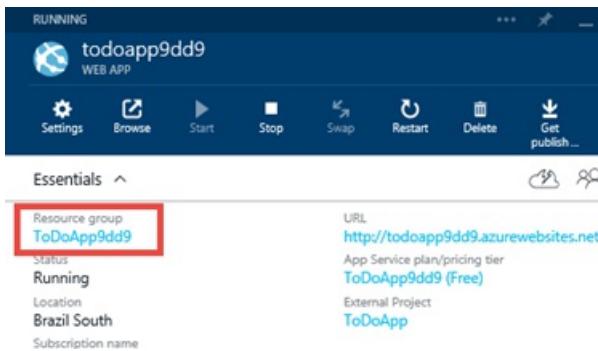
Directory	Subscription
Default Directory	Visual Studio Ultimate with MSDN
Resource Group	Resource Group Name
Create New	ToDoApp9dd9
Site Name <small>Name is available</small>	Site Location
ToDoApp9dd9	Brazil South
Sku	Sql Server Name
Free	todoapp9dd9-server
Sql Server Location	Sql Server Admin Login
East US 2	
Sql Server Admin Password	Sql Db Name
	DemosDB
Sql Db Collation	Sql Db Edition
SQL_Latin1_General_CI_AS	Web
Sql Db Max Size Bytes	Sql Db Service Objective Id
1073741824	910b4fcf-8a29-4c3e-958f-f7ba794388b2

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.

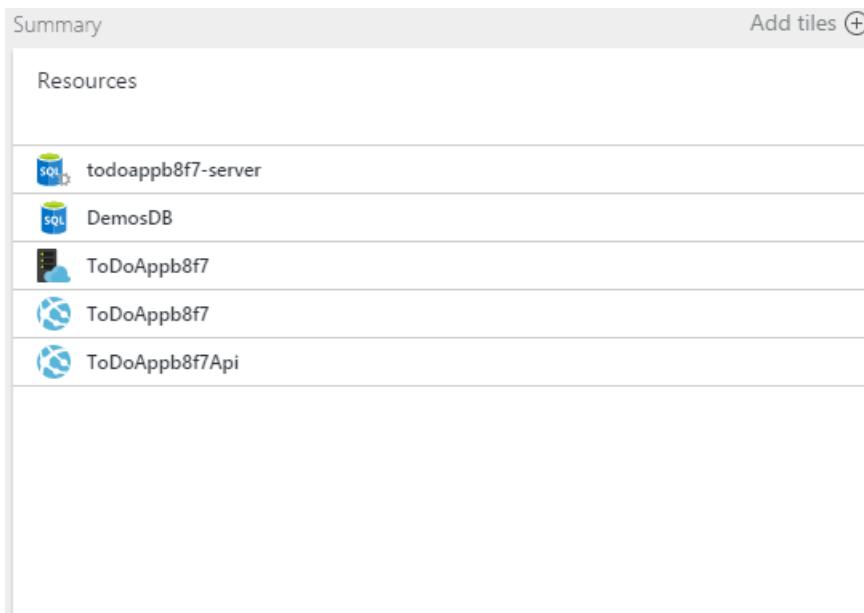


The UI would be a little slow when you first browse to it because the apps are just starting up, but convince yourself that it's a fully-functional application.

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**.



7. In the resource group blade, note that there are already two apps and one SQL Database in the resource group.



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:

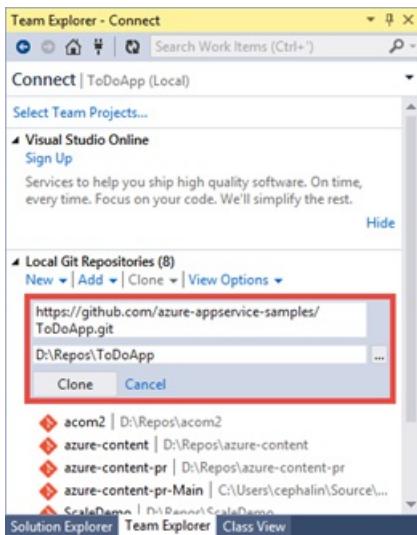
- The Deploy to Azure button
- azuredeploy.json in the repo root

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.

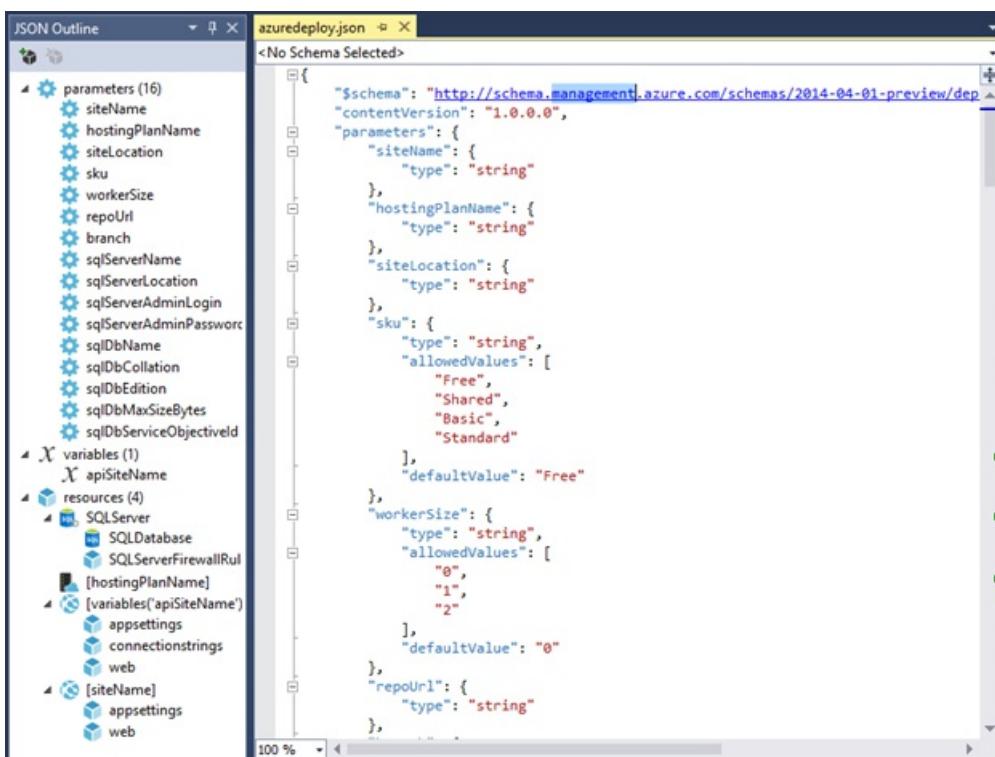
## Examine (or edit) AZUREDEPLOY.JSON

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.



- 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 `azuredeploy.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.

The screenshot shows the Azure portal's JSON Outline and code editor for a deployment template named 'azuredploy.json'. The JSON Outline on the left lists resources: parameters (16), variables (1), resources (4), SQLServer, SQLDatabase, SQLServerFirewallRule, HostingPlanName, [variables('apiSiteName')], [siteName]. The 'HostingPlanName' resource is selected and highlighted with a red box. The code editor on the right shows the corresponding JSON code for this resource.

```

    "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.

The screenshot shows the Azure portal's JSON Outline and code editor for a deployment template named 'azuredploy.json'. The JSON Outline on the left lists resources: parameters (16), variables (1), resources (4), SQLServer, SQLDatabase, SQLServerFirewallRule, [hostingPlanName], [variables('apiSiteName')], [siteName]. The 'SQLServer' resource is selected and highlighted with a red box. The code editor on the right shows the corresponding JSON code for this resource.

```

    "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")))" }
  }
},
```

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:

Azure Resource Explorer (Preview) Search Default Directory (cephaslin@hotmail.onmicrosoft.com)

- + Default-Storage-WestUS
- + Default-Web-CentralUS
- + Default-Web-EastUS
- + Default-Web-EastUS2
- + Default-Web-NorthCentralUS
- + Default-Web-NorthEurope
- + Default-Web-WestUS
- ToDoApp9dd9
  - providers
    - \* Show all
  - Microsoft.Sql
    - \* Show all
    - servers
      - + todoapp9dd9-server
  - Microsoft.Web
    - serverFarms
      - ToDoApp9dd9
    - sites
      - + ToDoApp9dd9
      - ToDoApp9dd9Api
        - config
  - appsettings
  - connectionstrings
  - metadata
  - publishingcredentials
  - web
  - slots
  - sourcecontrols

## appsettings

Data (GET, PUT) Actions (POST, DELETE) Create

Documentation

POST Edit https://management.azure.com/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/sites/ToDoApp9dd9Api/config/appsettings

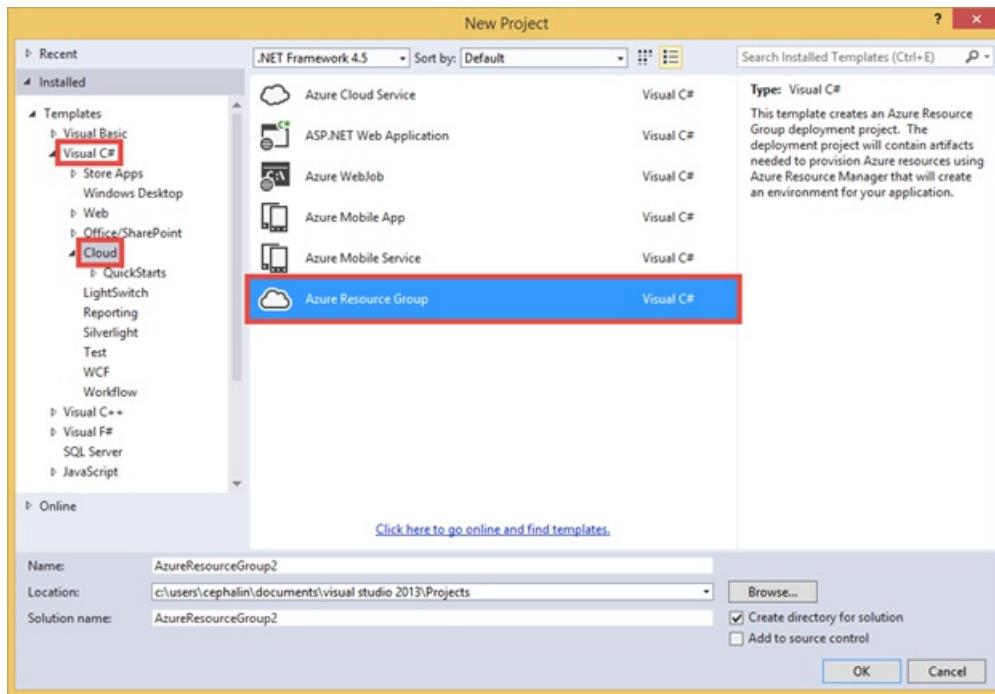
```
1 - {  
2   "id": "/subscriptions/62f3ac8c-ca8d-407b-abd8-04c5496b2221/resourceGroups/ToDoApp9dd9/providers/Microsoft.Web/sites/ToDoApp9dd9Api/config/appsettings",  
3   "name": "appsettings",  
4   "type": "Microsoft.Web/sites/config",  
5   "kind": null,  
6   "location": "Brazil South",  
7   "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}
```

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.

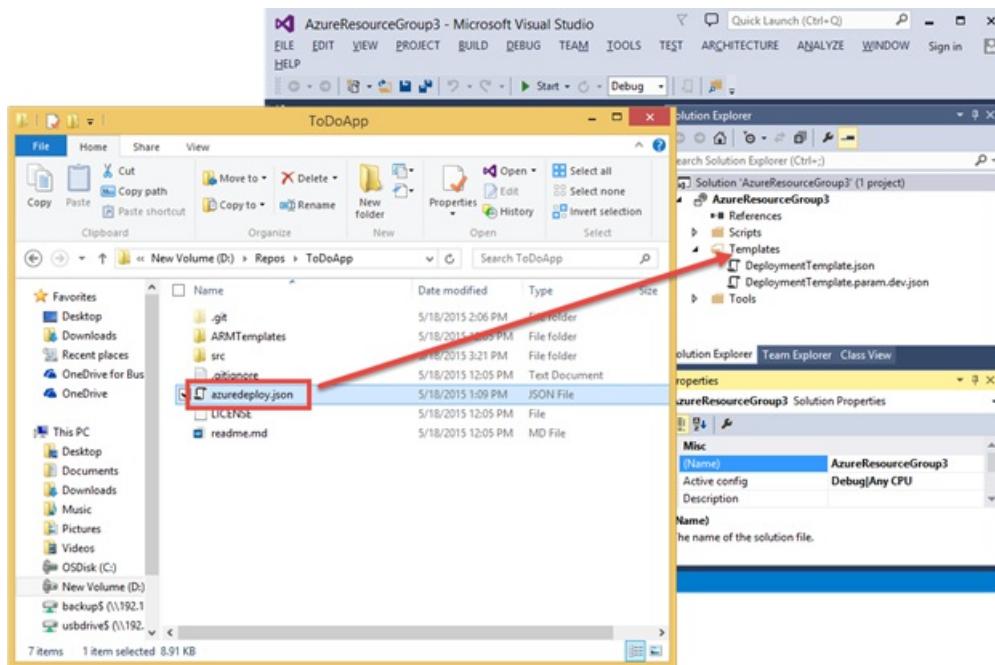
## Deploy the resource group template yourself

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:

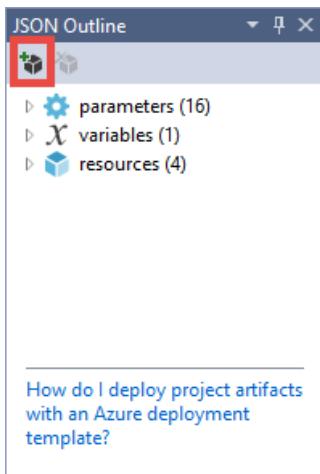
1. In Visual Studio, click **File > New > Project**.
2. Click **Visual C# > Cloud > Azure Resource Group**, then click **OK**.



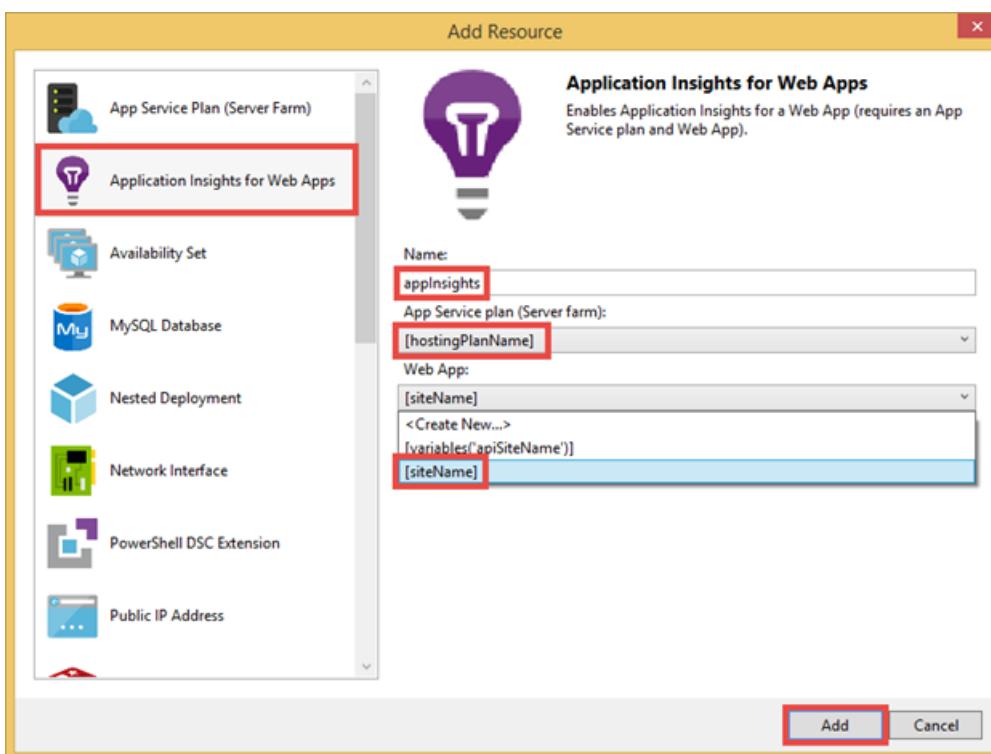
3. In **Select Azure Template**, select **Blank Template** and click **OK**.
4. Drag `azuredeploy.json` into the **Template** folder of your new project.



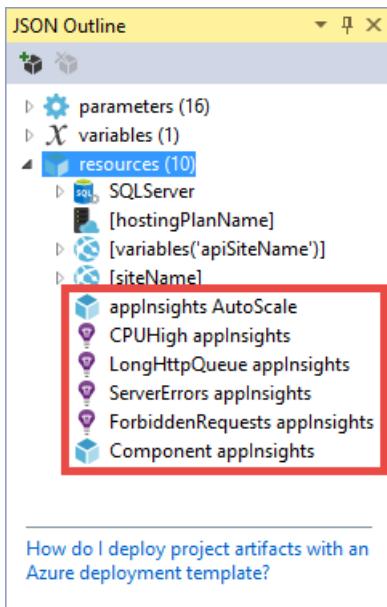
5. From Solution Explorer, open the copied `azuredetect.json`.
6. Just for the sake of the demonstration, let's add some standard Application Insight resources to our JSON file, by clicking **Add Resource**. If you're just interested in deploying the JSON file, skip to the deploy steps.



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 **appInsights 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.

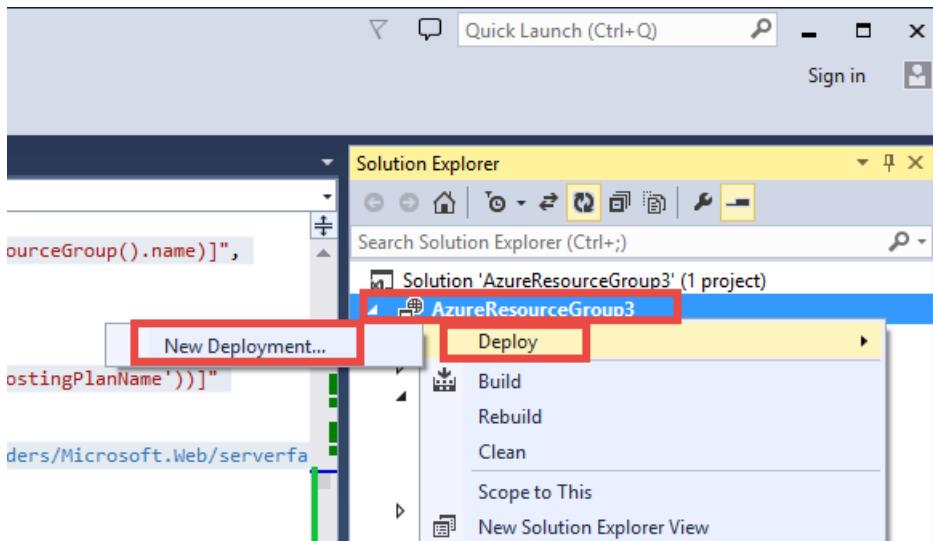
```
{
  "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
  "type": "Microsoft.Insights/autoscalesettings",
  "location": "[parameters('siteLocation')]",
  "apiVersion": "2014-04-01",
  "dependsOn": [...],
  "tags": [...],
  "properties": {
    "name": "[concat(parameters('hostingPlanName'), '-', resourceGroup().name)]",
    "profiles": [...],
    "enabled": true,
    "targetResourceUri": "[concat(resourceGroup().id, '/providers/Microsoft.W...")]
  }
},
```

10. In the JSON Outline, click **CPUHigh appInsights** to highlight its JSON code. This is an alert.

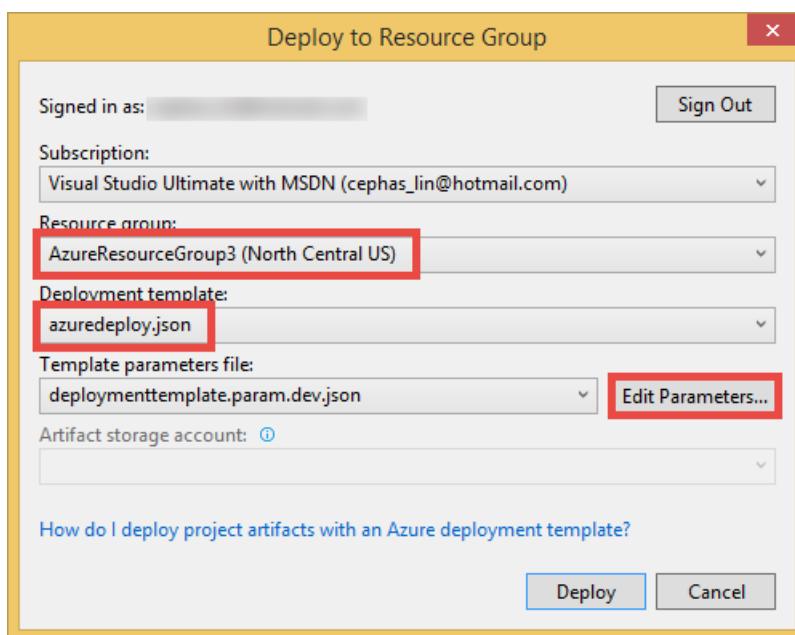
11. Locate the `location` and `isEnabled` properties and set them as shown below. Do the same for the other three alerts (purple bulbs).

```
{
  "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": [...]
  }
},
```

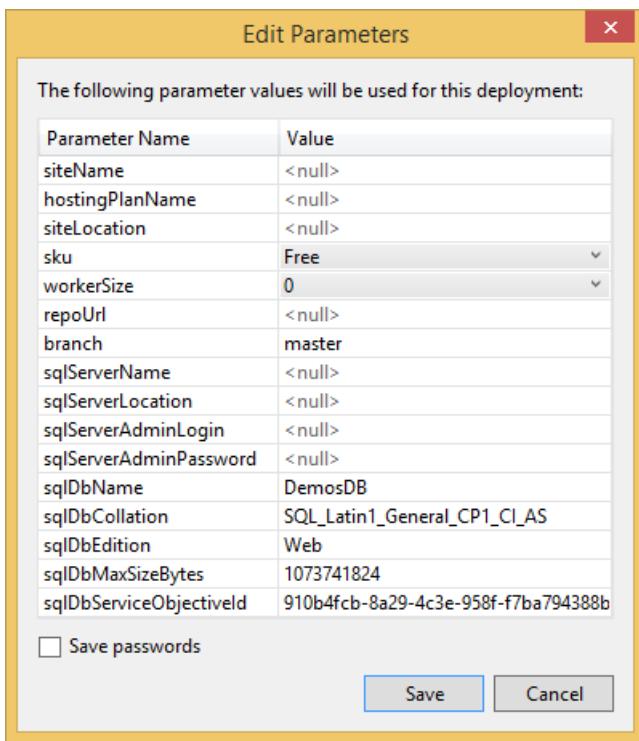
12. You're now ready to deploy. Right-click the project and select **Deploy > New Deployment**.



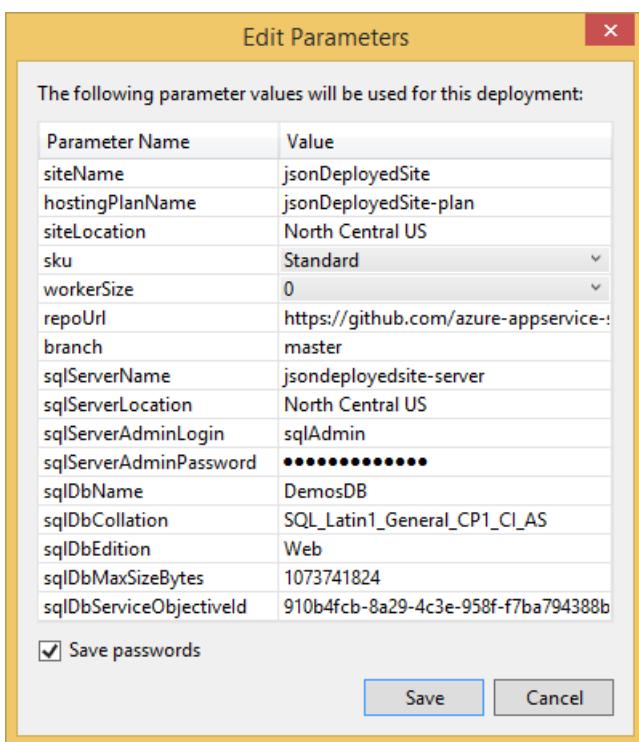
13. Log into your Azure account if you haven't already done so.
14. Select an existing resource group in your subscription or create a new one, select **azuredeploy.json**, and then click **Edit Parameters**.



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 {
    $AzureToolsUserAgentString = New-Object -TypeName System.Net.Http.Headers.ProductInfoHeaderValue -ArgumentList @(
        "User-Agent", "PowerShell/$($PSVersionTable.PSVersion.Major).$($PSVersionTable.PSVersion.Minor) ($($PSVersionTable.OSName))",
        "Accept", "application/json"
    )
    $AzureToolsUserAgentString.ToString()
}

# 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

1/28/2020 • 3 minutes to read • [Edit Online](#)

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.

## Configure user-level credentials

You can configure your user-level credentials in any app's [resource page](#). Regardless in which app you configure these credentials, it applies to all apps and for all subscriptions in your Azure account.

### In the Cloud Shell

To configure the deployment user in the [Cloud Shell](#), 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`. If you get a `'Conflict'. Details: 409` error, change the username. If you get a `'Bad Request'. Details: 400` error, use a stronger password.

### In the portal

In the Azure portal, you must have at least one app before you can access the deployment credentials page. To configure your user-level credentials:

1. In the [Azure portal](#), from the left menu, select **App Services** > <any\_app> > **Deployment center** > **FTP > Dashboard**.

Or, if you've already configured Git deployment, select **App Services > <any\_app> > Deployment center > FTP/Credentials**.

## 2. Select **User Credentials**, configure the user name and password, and then select **Save Credentials**.

Once you have set your deployment credentials, you can find the *Git* deployment username in your app's **Overview** page,

If Git deployment is configured, the page shows a **Git/deployment username**; otherwise, an **FTP/deployment username**.

### NOTE

Azure does not show your user-level deployment password. If you forget the password, you can reset your credentials by following the steps in this section.

## Use user-level credentials with FTP/FTPS

Authenticating to an FTP/FTPS endpoint using user-level credentials requires a username in the following format: <app-name>\<user-name>

Since user-level 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 and reset app-level credentials

To get the app-level credentials:

1. In the [Azure portal](#), from the left menu, select **App Services** > <any\_app> > **Deployment center** > **FTP/Credentials**.
2. Select **App Credentials**, and select the **Copy** link to copy the username or password.

To reset the app-level credentials, select **Reset Credentials** in the same dialog.

## 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

1/6/2020 • 17 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 top navigation bar, there is a search bar with the text "app services". Below the search bar, a list of results titled "Services" is displayed, with a total of "All 60 results". The first result, "App Services", is highlighted with a red box. Other results 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". To the left of the search results, there is a sidebar with sections for "Azure services" (Create a resource, Resource groups), "Recent resources" (myFirstAzureWebA, WebApplicationAS, cs4316e81020662x), and "Navigate" (Subscriptions). On the right side, there is a "LAST VIEWED" section showing items last viewed 1 h ago, 2 h ago, and 2 d ago.

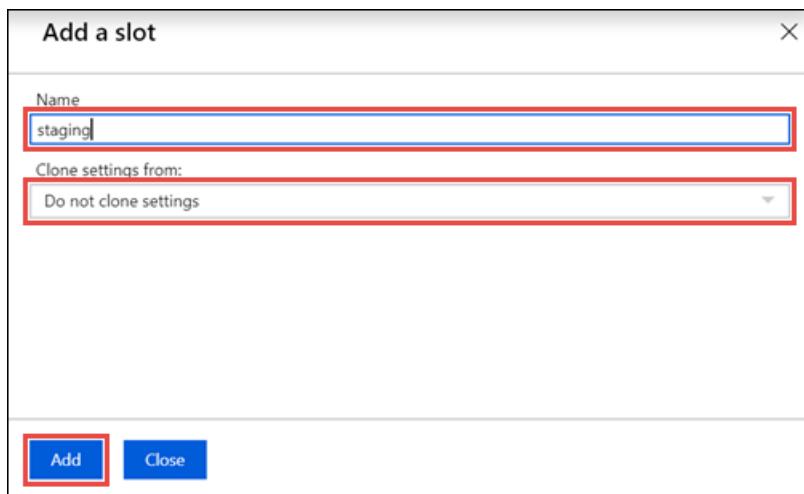
2. In the left pane, select **Deployment slots** > **Add Slot**.

The screenshot shows the "Deployment slots" blade for the app "my-demo-app". The left sidebar has tabs for Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, Deployment (Quickstart, Deployment slots, Deployment Center), Settings, and Configuration. The "Deployment slots" tab is selected and highlighted with a red box. The main area shows 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 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 the current deployment slot: NAME (my-demo-app), STATUS (Running), APP SERVICE PLAN (myAppServicePlan), and TRAFFIC % (100). The "PRODUCTION" status is highlighted with a green box. At the top of the blade, there are buttons for Save, Discard, Swap, Refresh, and a "+ Add Slot" button, which is also highlighted with 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.

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. The name of the slot is shown at the top of the page to remind you that you're viewing the deployment slot.

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.

## 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.

### 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 \*
- Virtual network integration \*
- Service endpoints \*
- Azure Content Delivery Network \*

Features marked with an asterisk (\*) are planned to be unswapped.

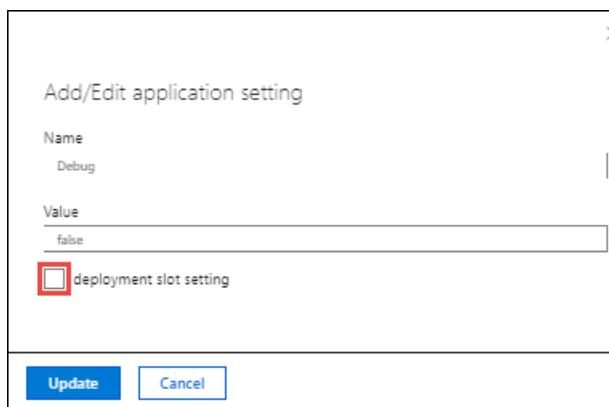
## Settings that aren't swapped:

- Publishing endpoints
- Custom domain names
- Non-public certificates and TLS/SSL settings
- Scale settings
- WebJobs schedulers
- IP restrictions
- Always On
- Diagnostic log settings
- Cross-origin resource sharing (CORS)

### NOTE

Certain app settings that apply to unswapped settings are also not swapped. For example, since diagnostic log 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.



## 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 Azure portal interface for managing deployment slots. On the left, there's a sidebar with various navigation options like 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. The main content area is titled 'Deployment Slots' and contains a table with two rows:

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 **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**.

The screenshot shows the 'Swap' dialog box. It has two dropdown menus: 'Source' set to 'my-demo-app-staging' and 'Target' set to 'my-demo-app' (labeled 'PRODUCTION'). Below these is a note: 'Swap with preview can only be used with sites that have deployment slot settings enabled'. There's a checkbox for 'Perform swap with preview'. The main area is titled 'Config Changes' with a sub-section 'Source Changes' highlighted with a red box. It lists a single configuration change for 'MyDbConnectionString':

SETTING	TYPE	OLD VALUE	NEW VALUE
MyDbConnectionString	ConnectionString	Server=tcp:stagingser... Info=False;User ID=... <username>;Passwor... <password>;Multiple... Timeout=30;	Server=tcp:productio... Info=False;User ID=... <username>;Passwor... <password>;Multiple... Timeout=30;

At the bottom are two buttons: 'Swap' (highlighted with a red box) and 'Close'.

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](#).

3. 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)

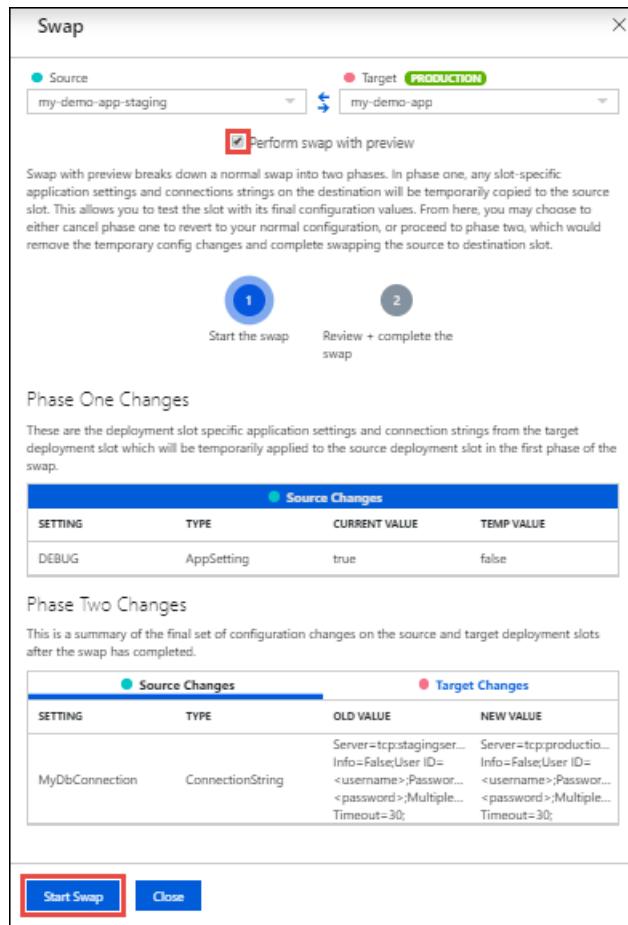
Before you swap into production as the target slot, validate that the app runs with the swapped settings. The source slot is also warmed up before the swap completion, which is desirable for mission-critical applications.

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:

1. 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.

Auto swap streamlines Azure DevOps scenarios where you want to deploy your app continuously with zero cold

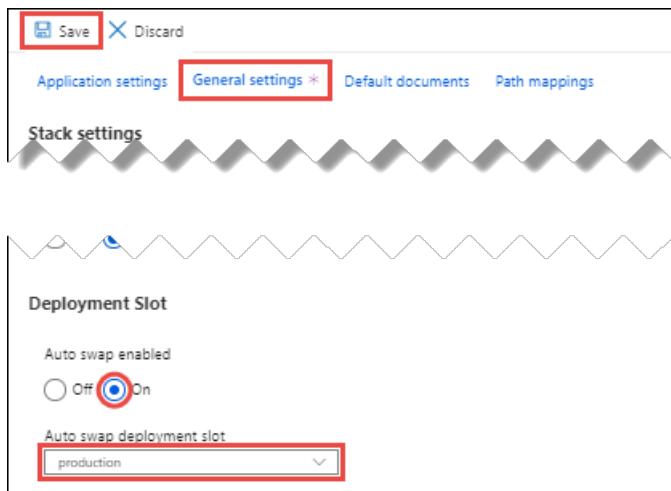
starts and zero downtime for customers of the app. When auto swap is enabled from a slot into production, every time you push your code changes to that slot, App Service automatically [swaps the app into production](#) after it's warmed up in the source slot.

#### 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.

#### 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 PRODUCTION	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

Next to the Azure portal, 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 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</a>
```

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.

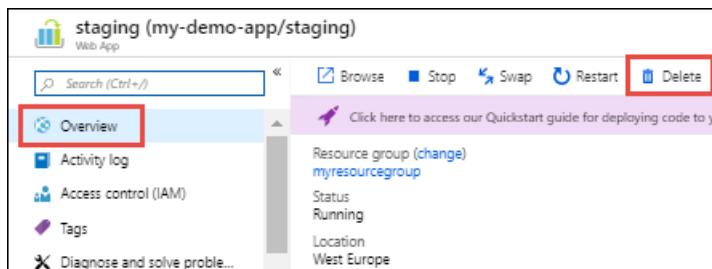
To let users opt in to your beta app, set the same query parameter to the name of the non-production slot. Here's an example:

```
<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**. Select **Delete** on the command bar.



## Automate with PowerShell

#### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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>
```

- Some [IP restriction rules](#) might prevent the swap operation from sending HTTP requests to your app. IPv4 address ranges that start with `10.` and `100.` are internal to your deployment. You should allow them to connect to your app.
- 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](#)

# Buy a custom domain name for Azure App Service

2/4/2020 • 9 minutes to read • [Edit Online](#)

App Service domains are top-level 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.
- [Remove the spending limit on your subscription](#). You cannot buy App Service domains with free subscription credits.

## Prepare 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.

To use custom domains in Azure App Service, your app's [App Service plan](#) must be a paid tier (**Shared**, **Basic**, **Standard**, or **Premium**). In this step, you make sure that the app is in the supported pricing tier.

### Sign in to Azure

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

### Navigate to the app in the Azure portal

From the left menu, select **App Services**, and then select the name of the app.

Home > App Services

## App Services

Microsoft

+ Add Edit columns Refresh Assign tags Start Restart More

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

Filter by na... All subsc... All resou... All locati... All tags No group...

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 see the management page of the App Service app.

### Check the pricing tier

In the left navigation of the app page, scroll to the **Settings** section and 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 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

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

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](#).

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

&lt;price&gt;/Month (Estimated)

**B1**

100 total ACU

1.75 GB memory

A-Series compute equivalent

&lt;price&gt;/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**

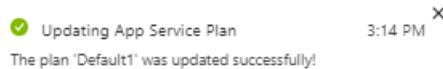
Memory per instance available to run applications deployed and running in...

**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.



## Buy the domain

### Pricing Information

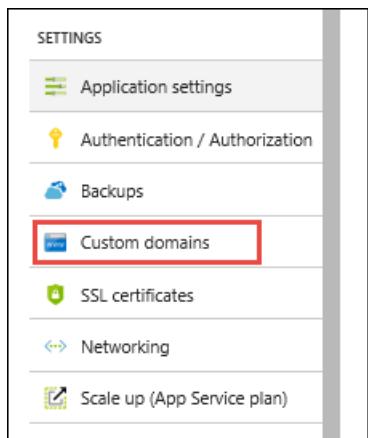
For pricing information on Azure App Service Domains, visit the [App Service Pricing page](#) and scroll down to App Service Domain.

[Sign in to Azure](#)

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

## Launch Buy domains

In the **App Services** tab, click the name of your app, select **Settings**, and then select **Custom domains**



In the **Custom domains** page, click **Buy Domain**.

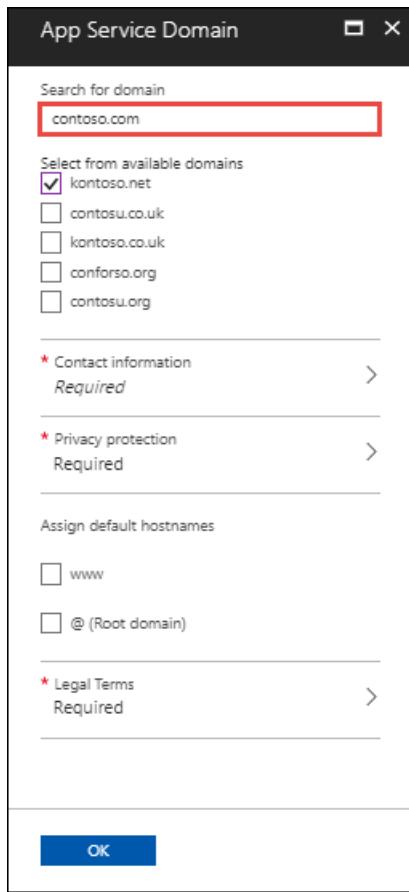
A screenshot of the 'App Service Domains' page. At the top, there's a 'WWW' icon and the title 'App Service Domains'. Below that, a message says 'Purchase and manage domains for your Azure services with auto-renew and privacy protection.' with a 'Learn more' link. In the center, there's a 'Buy Domain' button with a plus sign icon, which is highlighted with a red box. Below the button is a table with three columns: 'DOMAINS', 'EXPIRES', and 'STATUS'. There is one row in the table showing a domain that expires on 'March 1, 2018, 10:35:00 AM GMT+1' and is currently 'Active'.

### NOTE

If you cannot see the **App Service Domains** section, you need to remove the spending limit on your Azure account (see [Prerequisites](#)).

## Configure the domain purchase

In the **App Service Domain** page, in the **Search for domain** box, type the domain name you want to buy and type **Enter**. The suggested available domains are shown just below the text box. Select one or more domains you want to buy.



#### 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.*

Click the **Contact Information** and fill out the domain's contact information form. When finished, click **OK** to return to the App Service Domain page.

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 purchase domains.

Next, select the desired options for your domain. See the following table for explanations:

SETTING	SUGGESTED VALUE	DESCRIPTION
Privacy protection	Enable	Opt in to "Privacy protection", which is included in the purchase price <i>for free</i> . Some top-level domains are managed by registrars that do not support privacy protection, and they are listed on the <b>Privacy protection</b> page.
Assign default hostnames	<b>www</b> and <b>@</b>	Select the desired hostname bindings, if desired. When the domain purchase operation is complete, your app can be accessed at the selected hostnames. If the app is behind <a href="#">Azure Traffic Manager</a> , you don't see the option to assign the root domain (@), because Traffic Manager does not support A records. You can make changes to the hostname assignments after the domain purchase completes.

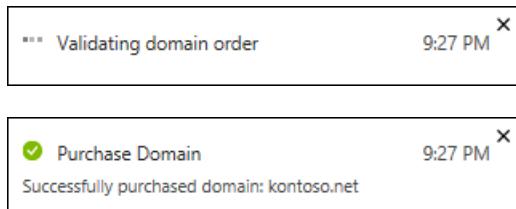
## Accept terms and purchase

Click **Legal Terms** to review the terms and the charges, then click **Buy**.

### NOTE

App Service Domains use GoDaddy for domain registration and Azure DNS to host the domains. In addition to the domain registration fee, usage charges for Azure DNS apply. For information, see [Azure DNS Pricing](#).

Back in the **App Service Domain** page, click **OK**. While the operation is in progress, you see the following notifications:



## Test the hostnames

If you have assigned default hostnames to your app, you also see a success notification for each selected hostname.



You also see the selected hostnames in the **Custom domains** page, in the **Custom Hostnames** section.

The screenshot shows the 'Custom Domains' blade in the Azure portal. At the top, there's a globe icon and the title 'Custom Domains'. Below that is a sub-header 'Configure and manage custom domains assigned to your app' with a 'Learn more' link. Underneath are input fields for 'IP address' (set to 13.69.68.6) and 'HTTPS Only' (set to Off). A large blue '+' button labeled 'Add custom domain' is present. A status filter bar shows 'All (3)', 'Not Secure (2)', and 'Secure (1)'. The main table lists assigned custom domains with columns for 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. The 'Not Secure' rows for contoso.com and www.contoso.com have red exclamation marks and 'Add binding' buttons. The 'Secure' row for my-demo-app.azurewebsites.net has a green checkmark and a similar 'Add binding' button. Each row also has a '...' button.

SSL STATE	ASSIGNED CUSTOM DOMAINS	SSL BINDING
! Not Secure	contoso.com	Add binding
! Not Secure	www.contoso.com	Add binding
✓ Secure	my-demo-app.azurewebsites.net	

### NOTE

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

To test the hostnames, navigate to the listed hostnames in the browser. In the example in the preceding screenshot, try navigating to *kontoso.net* and *www.kontoso.net*.

## Assign hostnames to app

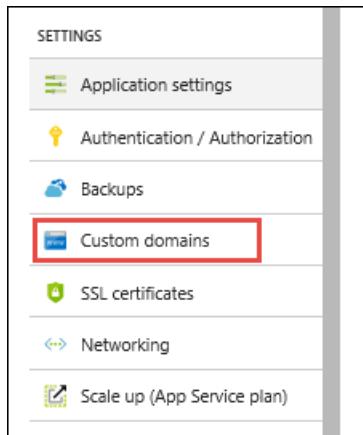
If you choose not to assign one or more default hostnames to your app during the purchase process, or if you need to assign a hostname not listed, you can assign a hostname anytime.

You can also assign hostnames in the App Service Domain to any other app. The steps depend on whether the App Service Domain and the app belong to the same subscription.

- Different subscription: Map custom DNS records from the App Service Domain to the app like an externally purchased domain. For information on adding custom DNS names to an App Service Domain, see [Manage custom DNS records](#). To map an external purchased domain to an app, see [Map an existing custom DNS name to Azure App Service](#).
- Same subscription: Use the following steps.

### Launch add hostname

In the **App Services** page, select the name of your app that you want to assign hostnames to, select **Settings**, and then select **Custom domains**.



Make sure that your purchased domain is listed in the **App Service Domains** section, but don't select it.

App Service Domains		
DOMAINS	EXPIRES	STATUS
kontoso.net	July 19, 2018 2:27:28 PM CEST	Active

#### NOTE

All App Service Domains in the same subscription are shown in the app's **Custom domains** page. If your domain is in the app's subscription, but you cannot see it in the app's **Custom domains** page, try reopening the **Custom domains** page or refresh the webpage. Also, check the notification bell at the top of the Azure portal for progress or creation failures.

Select **Add hostname**.

### Configure hostname

In the **Add hostname** dialog, type the fully qualified domain name of your App Service Domain or any subdomain. For example:

- kontoso.net
- www.kontoso.net
- abc.kontoso.net

When finished, select **Validate**. The hostname record type is automatically selected for you.

Select **Add hostname**.

When the operation is complete, you see a success notification for the assigned hostname.



## Close add hostname

In the **Add hostname** page, assign any other hostname to your app, as desired. When finished, close the **Add hostname** page.

You should now see the newly assigned hostname(s) in your app's **Custom domains** page.

HOSTNAMES ASSIGNED TO SITE	
abc.kontoso.net	...
kontoso.net	...
www.kontoso.net	...
webapp-custom-dns.azurewebsites.net	...

## Test the hostnames

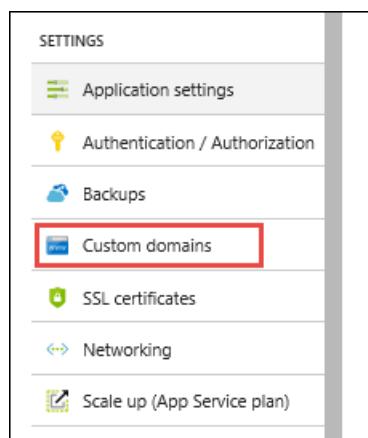
Navigate to the listed hostnames in the browser. In the example in the preceding screenshot, try navigating to *abc.kontoso.net*.

## 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.

In the **App Services** tab, click the name of your app, select **Settings**, and then select **Custom domains**.



In the **App Service Domains** section, select the domain you want to configure.

App Service Domains		
DOMAINS	EXPIRES	STATUS
kontoso.net	July 19, 2018 2:27:28 PM CEST	Active

From the left navigation of the domain, select **Domain renewal**. To stop renewing your domain automatically, select **Off**, and then **Save**.

The screenshot shows the Azure portal interface for managing an App Service Domain named 'kontoso.net'. On the left, a navigation menu lists various settings like Overview, Access control (IAM), Tags, Properties, Locks, Automation script, Hostname bindings, Domain renewal (which is selected and highlighted with a red box), and DNS zone. The main content area is titled 'Domain renewal' and contains a note: 'App Service domains can be set to auto renew to prevent expiration and unexpected domain ownership loss.' Below this is a switch labeled 'Auto renew domain:' with options 'On' and 'Off' (which is selected). There are 'Save' and 'Discard' buttons. A warning message box states: '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.' At the bottom, the 'Expiration date:' is listed as 'November 27, 2018 9:04:20 AM CET' and there is a 'Renew domain' button.

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 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.

### Open App Service Domain

In the Azure portal, from the left menu, select **All services > App Service Domains**.

The screenshot shows the Microsoft Azure portal's search interface. The search bar at the top contains the text "Search resources, services and docs". Below it, a search result card for "app service" is displayed. The card has a title "All services" and a subtitle "app service". It lists several items under "App Services": "App Service Certificates", "App Service Domains", "App Service Environments", "App Service plans", "App Services" (with a note "Keywords: web apps"), "Citrix XenDesktop Essentials" (with a note "Keywords: VDI appliance"), "Cloud services (classic)" (with a note "Keywords: apps"), and "Service catalog managed application definitions". Each item has a star icon to its right.

Select the domain to manage.

#### Access DNS zone

In the domain's left menu, select **DNS zone**.

The screenshot shows the Azure portal's "kontoso.net App Service Domain" page. The left sidebar includes a search bar and links for "Automation script", "Hostname bindings", "Domain renewal", and "DNS zone", with "DNS zone" highlighted by a red box. The main pane displays "Advanced management" and "Cancel purch" buttons. Under "Essentials", it shows the "Resource group" as "myResourceGroup", "Status" as "Active", "Location" as "global", "Subscription name" (redacted), and "Subscription ID" (redacted).

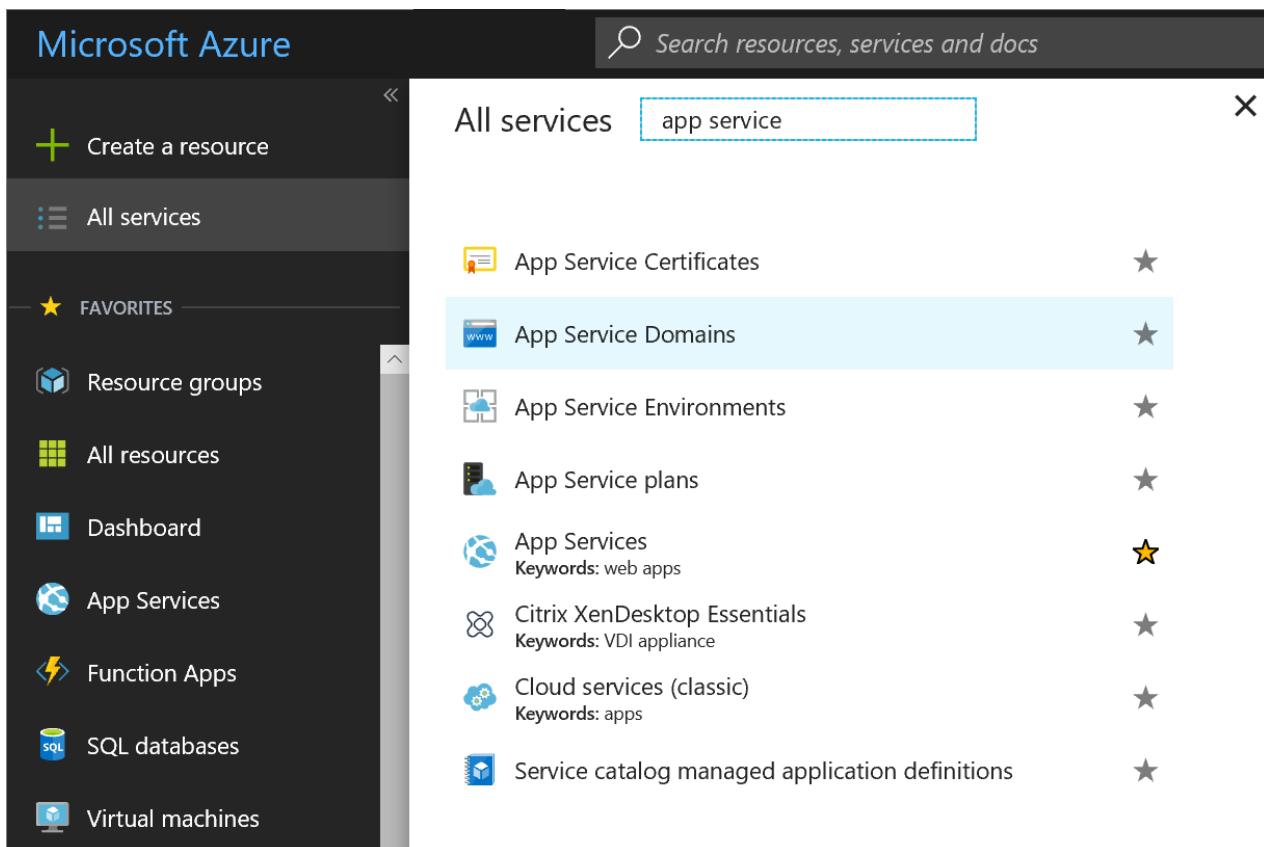
This action opens the [DNS zone](#) page of your App Service Domain in Azure DNS. For information on how to edit DNS records, see [How to manage DNS Zones in the Azure portal](#).

#### 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.

#### Open App Service Domain

In the Azure portal, from the left menu, select **All services > App Service Domains**.

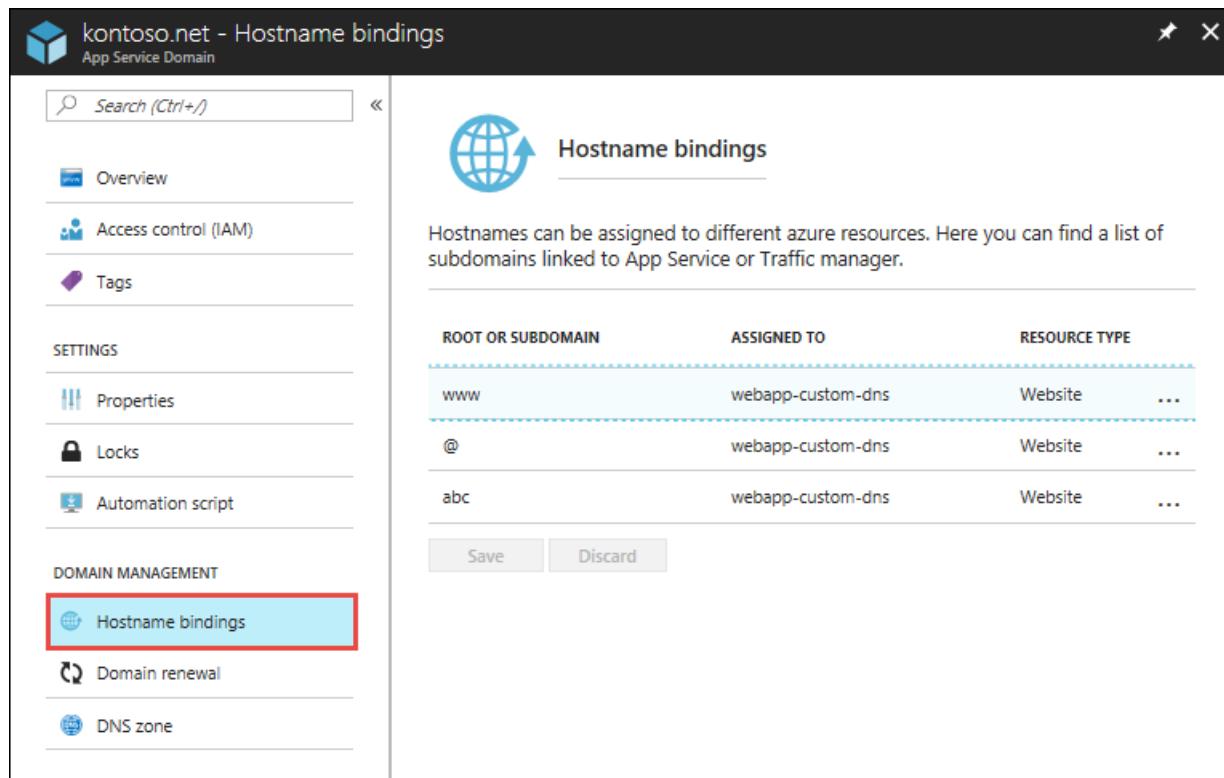


The screenshot shows the Microsoft Azure search interface. The search bar at the top contains the text "Search resources, services and docs". Below the search bar, the text "All services" is displayed, followed by a search input field containing "app service". A list of services is shown, each with a star icon to its right. The services listed are: App Service Certificates, App Service Domains (which is highlighted with a blue background), App Service Environments, App Service plans, App Services (with a note "Keywords: web apps"), Citrix XenDesktop Essentials (with a note "Keywords: VDI appliance"), Cloud services (classic) (with a note "Keywords: apps"), and Service catalog managed application definitions.

Select the domain to you want to cancel or delete.

### Delete hostname bindings

In the domain's left menu, select **Hostname bindings**. The hostname bindings from all Azure services are listed here.



The screenshot shows the Azure portal interface for the domain "kontoso.net - Hostname bindings". The left sidebar includes options like Overview, Access control (IAM), Tags, Properties, Locks, Automation script, and Domain management. Under Domain management, the "Hostname bindings" option is highlighted with a red box. The main content area is titled "Hostname bindings" and contains the following text: "Hostnames can be assigned to different azure resources. Here you can find a list of subdomains linked to App Service or Traffic manager." Below this is a table showing the list of hostname bindings:

ROOT OR SUBDOMAIN	ASSIGNED TO	RESOURCE TYPE	...
www	webapp-custom-dns	Website	...
@	webapp-custom-dns	Website	...
abc	webapp-custom-dns	Website	...

At the bottom of the table are "Save" and "Discard" buttons.

You cannot delete the App Service Domain until all hostname bindings are deleted.

Delete each hostname binding by selecting ... > **Delete**. After all the bindings are deleted, select **Save**.

The screenshot shows the 'Hostname bindings' section in the Azure portal. It lists two entries:

ROOT OR SUBDOMAIN	ASSIGNED TO	RESOURCE TYPE
www	webapp-custom-dns	Website
@	webapp-custi	... delete ...

At the bottom are 'Save' and 'Discard' buttons. A red box highlights the 'delete' button in the third row.

## Cancel or delete

In the domain's left menu, select **Overview**.

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 'kontoso.net' domain overview page. The left sidebar includes 'Search (Ctrl+/)', 'Overview' (which is selected and highlighted in blue), 'Access control (IAM)', and 'Tags'. The main content area shows:

Resource group	Domain
myResourceGroup	http://kontoso.net
Status	Expiration date
Active	July 19, 2018 2:27:28 P

A red box highlights the 'Cancel purchase' button in the top right corner.

To confirm the operation, select **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 [Direct default URL to a custom directory](#).

# Configuring a custom domain name for a web app in Azure App Service using Traffic Manager

12/2/2019 • 7 minutes to read • [Edit Online](#)

When you use a Microsoft Azure Traffic Manager to load balance traffic to your Azure Website, that website can then be accessed using the **\*.trafficmanager.net** domain name assigned by Azure. You can also associate a custom domain name, such as [www.contoso.com](#), with your website in order to provide a more recognizable domain name for your users.

This article provides generic instructions for using a custom domain name with an [App Service](#) app that is integrated with [Traffic Manager](#) for load balancing.

If you do not already have a Traffic Manager profile, use the information in [Create a Traffic Manager profile using Quick Create](#) to create one. Note the **.trafficmanager.net** domain name associated with your Traffic Manager profile, as this will be used later by later steps in this document.

This article is for Azure App Service (Web Apps, API Apps, Mobile Apps, Logic Apps); for Cloud Services, see [Configuring a custom domain name for an Azure cloud service](#).

## NOTE

If your app is load-balanced by [Azure Traffic Manager](#), click the selector at the top of this article to get specific steps.

**Custom domain names are not enabled for Free tier.** You must [scale up to a higher pricing tier](#), which may change how much you are billed for your subscription. See [App Service Pricing](#) for more information.

## Understanding DNS records

The Domain Name System (DNS) is used to locate things on the internet. For example, when you enter an address in your browser, or click a link on a web page, it uses DNS to translate the domain into an IP address. The IP address is sort of like a street address, but it's not very human friendly. For example, it is much easier to remember a DNS name like **contoso.com** than it is to remember an IP address such as 192.168.1.88 or 2001:0:4137:1f67:24a2:3888:9cce:fea3.

The DNS system is based on *records*. Records associate a specific *name*, such as **contoso.com**, with either an IP address or another DNS name. When an application, such as a web browser, looks up a name in DNS, it finds the record, and uses whatever it points to as the address. If the value it points to is an IP address, the browser will use that value. If it points to another DNS name, then the application has to do resolution again. Ultimately, all name resolution will end in an IP address.

When you create an Azure Website, a DNS name is automatically assigned to the site. This name takes the form of **<yoursitename>.azurewebsites.net**. When you add your website as an Azure Traffic Manager endpoint, your website is then accessible through the **<yourtrafficmanagerprofile>.trafficmanager.net** domain.

## NOTE

When your website is configured as a Traffic Manager endpoint, you will use the **.trafficmanager.net** address when creating DNS records.

You can only use CNAME records with Traffic Manager

There are also multiple types of records, each with their own functions and limitations, but for websites configured to as Traffic Manager endpoints, we only care about one; **CNAME** records.

### CNAME or Alias record

A CNAME record maps a *specific* DNS name, such as **mail.contoso.com** or **www.contoso.com**, to another (canonical) domain name. In the case of Azure Websites using Traffic Manager, the canonical domain name is the **<myapp>.trafficmanager.net** domain name of your Traffic Manager profile. Once created, the CNAME creates an alias for the **<myapp>.trafficmanager.net** domain name. The CNAME entry will resolve to the IP address of your **<myapp>.trafficmanager.net** domain name automatically, so if the IP address of the website changes, you do not have to take any action.

Once traffic arrives at Traffic Manager, it then routes the traffic to your website, using the load balancing method it is configured for. This is completely transparent to visitors to your website. They will only see the custom domain name in their browser.

#### NOTE

Some domain registrars only allow you to map subdomains when using a CNAME record, such as **www.contoso.com**, and not root names, such as **contoso.com**. For more information on CNAME records, see the documentation provided by your registrar, the [Wikipedia entry on CNAME record](#), or the [IETF Domain Names - Implementation and Specification](#) document.

## Configure your web apps for standard mode

Setting a custom domain name on a web app that is integrated with Traffic Manager is only available for the **Standard** pricing tier.

For more information on the App Service pricing tiers, including how to change your app's pricing tier, see [Scale up an app in Azure](#).

## Add a DNS record for your custom domain

#### NOTE

If you have purchased domain through Azure App Service Web Apps then skip following steps and refer to the final step of [Buy Domain for Web Apps](#) article.

To associate your custom domain with a web app in Azure App Service, you must add a new entry in the DNS table for your custom domain. You do this by using the management tools from your domain provider.

#### NOTE

You can use Azure DNS to configure a custom DNS name for your Azure Web Apps. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

Sign in to the website of your domain provider.

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 is 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, you 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* your custom domain name (such as **contoso.com**) *to* the Traffic Manager domain name (**contoso.trafficmanager.net**) that is integrated with your web 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 web app, create a CNAME record from **awverify.www** to **contoso.trafficmanager.net**. You can then add "www.contoso.com" to your Web App without changing the "www" CNAME record. For more information, see [Create DNS records for a web app in a custom domain](#).

Once you have finished adding or modifying DNS records at your domain provider, save the changes.

## Enable Traffic Manager

After the records for your domain name have propagated, you should be able to use your browser to verify that your custom domain name can be used to access your web app in Azure App Service.

#### NOTE

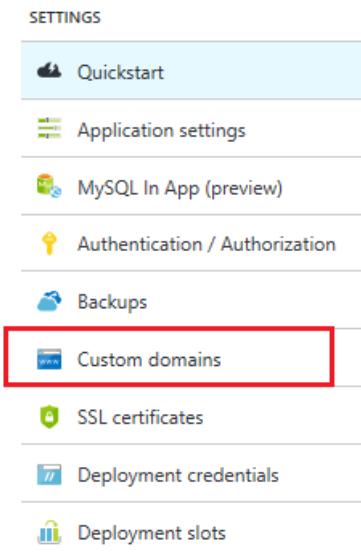
It can take some time for your CNAME to propagate through the DNS system. You can use a service such as <https://www.digwebinterface.com/> to verify that the CNAME is available.

If you have not already added your web app as a Traffic Manager endpoint, you must do this before name resolution will work, as the custom domain name routes to Traffic Manager. Traffic Manager then routes to your web app. Use the information in [Add or Delete Endpoints](#) to add your web app as an endpoint in your Traffic Manager profile.

#### NOTE

If your web app is not listed when adding an endpoint, verify that it is configured for **Standard** App Service plan mode. You must use **Standard** mode for your web app in order to work with Traffic Manager.

1. In your browser, open the [Azure Portal](#).
2. In the **Web Apps** tab, click the name of your web app, select **Settings**, and then select **Custom domains**



3. In the **Custom domains** blade, click **Add hostname**.
4. Use the **Hostname** text boxes to enter the custom domain name to associate with this web app.

The screenshot shows the 'Custom domains' blade. In the 'Hostnames' section, the 'Add hostname' button is highlighted with a red box. To the right, a modal dialog is open with a 'Hostname' input field containing 'www.contoso.com' and a 'Validate' button, which is also highlighted with a red box.

5. Click **Validate** to save the domain name configuration.
6. Upon clicking **Validate** Azure will kick off Domain Verification workflow. This will check for Domain ownership as well as Hostname availability and report success or detailed error with prescriptive guidance on how to fix the error.
7. Upon successful validation **Add hostname** button will become active and you will be able to assign the hostname. Now navigate to your custom domain name in a browser. You should now see your app running using your custom domain name.

Once configuration has completed, the custom domain name will be listed in the **domain names** section of your web app.

At this point, you should be able to enter the Traffic Manager domain name in your browser and see that it successfully takes you to your web app.

## Next steps

For more information, see the [Node.js Developer Center](#).

# Migrate an active DNS name to Azure App Service

12/2/2019 • 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 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 your Azure Web Apps. For more information, see [Use Azure DNS to provide custom domain settings for an Azure service](#).

Sign in to the website of your domain provider.

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 is 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, you 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.

### Create domain verification record

To verify domain ownership, Add a TXT record. The TXT record maps from *awverify.<subdomain>* to *<appname>.azurewebsites.net*.

The TXT record you need depends on the DNS record you want to migrate. For examples, see the following table (@ typically represents the root domain):

DNS RECORD EXAMPLE	TXT HOST	TXT VALUE
@ (root)	<i>awverify</i>	<i>&lt;appname&gt;.azurewebsites.net</i>
www (sub)	<i>awverify.www</i>	<i>&lt;appname&gt;.azurewebsites.net</i>
* (wildcard)	<i>awverify.*</i>	<i>&lt;appname&gt;.azurewebsites.net</i>

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

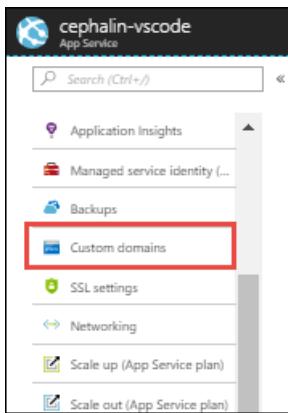
For certain providers, such as CloudFlare, `awverify.*` is not a valid record. Use `*` only instead.

### 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**.



In the **Custom domains** page, select the + icon next to **Add hostname**.

A screenshot of the 'Custom Domains' page. At the top, there's a globe icon and the title 'Custom Domains'. Below that, a sub-header says 'Configure and manage custom domains assigned to your app' with a 'Learn more' link. There are two input fields: 'IP address:' with '13.69.68.6' and 'HTTPS Only:' with a toggle switch set to 'Off'. A large blue '+' button is labeled 'Add custom domain'. Below this is a 'Status Filter' with 'All (1)', 'Not Secure (0)', and 'Secure (1)'. A table follows, with columns 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. One row is shown: a green checkmark under 'SSL STATE', 'my-demo-app.azurewebsites.net' under 'ASSIGNED CUSTOM DOMAINS', and an empty field under 'SSL BINDING'.

Type the fully qualified domain name that you added the TXT record for, such as [www.contoso.com](http://www.contoso.com). For a wildcard domain (like \*.contoso.com), you can use any DNS name that matches the wildcard domain.

Select **Validate**.

The **Add hostname** button is activated.

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. It starts with a header 'Add custom domain' and 'my-demo-app'. The first section is 'Custom domain' with an input field containing 'www.contoso.com' which is highlighted with a red box. Below it is a blue '+' button. The next section is 'Hostname record type' with a dropdown menu showing 'CNAME (www.example.com or any subdomain)' which is also highlighted with a red box. A 'CNAME configuration' section follows, featuring a globe icon and a sub-header 'CNAME configuration'. It contains a note: 'A CNAME record is used to specify that a domain name is an alias for another domain.' with a 'Learn More' link. There's a 'CNAME' input field with 'my-demo-app.azurewebsites.net' and a final 'Add custom domain' button which is also highlighted with a red box.

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. At the top, there's a status bar with 'IP address: 13.69.68.6' and an 'HTTPS Only' toggle switch set to 'Off'. Below this is a button to 'Add custom domain'. A 'Status Filter' dropdown shows 'All (2) Not Secure (1) Secure (1)'. The main table has columns for 'SSL STATE', 'ASSIGNED CUSTOM DOMAINS', and 'SSL BINDING'. It lists two entries: one 'Not Secure' entry for 'www.contoso.com' with an 'Add binding' button, and one 'Secure' entry for 'my-demo-app.azurewebsites.net'. There are also three vertical ellipsis buttons for more options.

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.

Close the **Add hostname** page by selecting **X** in the upper-right corner.

In the **Custom domains** page, copy the app's IP address.

The screenshot shows the 'Custom Hostnames' section of the Azure portal. It has a similar layout to the 'Custom Domains' page, with fields for 'IP address' (40.118.102.46) and 'HTTPS Only' (set to 'Off'). A 'HOSTNAMES ASSIGNED TO SITE' table lists 'cephalin-vscode.azurewebsites.net'. The 'SSL BINDING' column for this entry is empty.

### 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

FQDN EXAMPLE	RECORD TYPE	HOST	VALUE
*.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.

## Active domain in Azure

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 SSL certificate to App Service.

[Bind an SSL certificate to Azure App Service](#)

# Add an SSL certificate in Azure App Service

2/17/2020 • 16 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](#).

The following table lists the options you have for adding certificates in App Service:

OPTION	DESCRIPTION
Create a free App Service Managed Certificate (Preview)	A private certificate that's easy to use if you just need to secure your <a href="#">www</a> <a href="#">custom domain</a> or any non-naked domain 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

To follow this how-to guide:

- [Create an App Service app](#).
- Free certificate only: map a subdomain (for example, [www.contoso.com](#)) to App Service with a [CNAME record](#).

## Private certificate requirements

### NOTE

Azure Web Apps does **not** support AES256 and all pfx files should be encrypted with TripleDES.

The [free App Service Managed Certificate](#) or 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](#)
- Contains private key at least 2048 bits long

- Contains all intermediate certificates in the certificate chain

To secure a custom domain in an SSL 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 bind a custom SSL certificate (a third-party certificate or App Service certificate) to your web 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 visible, with 'App Services' highlighted with a red box. Other options 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 icons for 'Create a resource', 'App Services', 'Azure Database for PostgreSQL', and 'Azure Cosmos DB'. Below the sidebar, there's a 'Recent resources' section showing 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.

Home > App Services

## App Services

Microsoft

Add Edit columns Refresh Assign tags Start Restart More

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

Filter by na... All subsc... All resou... All locati... All tags No group...

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)**.

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)

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...

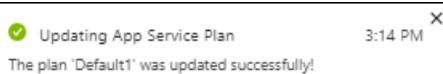


#### Storage

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.



### Create a free certificate (Preview)

The free App Service Managed Certificate is a turn-key solution for securing your custom DNS name in App Service. It's a fully functional SSL certificate that's managed by App Service and renewed automatically. The free certificate comes with the following limitations:

- Does not support wildcard certificates.

- Does not support naked domains.
- Is not exportable.
- Does not support DNS A-records.

#### NOTE

The free certificate is issued by DigiCert. For some top-level domains, you must explicitly allow DigiCert as a certificate issuer by creating a [CAA domain record](#) with the value: `0 issue digicert.com`.

To create a free App Service Managed Certificate:

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, the 'TLS/SSL settings' option is highlighted with a red box. In the center, the 'Private Key Certificates (.pfx)' tab is selected. Below it, the 'Private Key Certificate' section is shown with a 'PFX' icon. There are four buttons for actions: 'Import App Service Certificate', 'Upload Certificate', 'Import Key Vault Certificate', and 'Create App Service Managed Certificate'. The 'Create App Service Managed Certificate' button is also highlighted with a red box. The status filter at the bottom is set to 'All'.

Any non-naked domain that's properly mapped to your app with a CNAME record is listed in the dialog. 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.

The screenshot shows the 'Private Key Certificates' list. The status filter is set to 'All'. A single certificate is listed: 'www.contoso.com' is healthy, expiration is 4/11/2020, and the thumbprint is 6A3BCCA5CC4B0158F0A097CE9F39... . An ellipsis (...) button is also present.

#### 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](#)).
- Synchronizes 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.

## Start certificate order

Start an App Service certificate order in the [App Service Certificate create page](#).

The screenshot shows two side-by-side windows. The left window is titled 'App Service Certificate' and contains fields for Name (ContosoCert), Naked Domain Host Name (contosocertificate.com), Subscription (selected), Resource Group (contosocertdemo), and Certificate SKU (Standard). The right window is titled 'Certificate Pricing' and compares two SKUs: 'S1 Standard' at \$69.99 USD/YEAR (ESTIMATED) and 'W1 Wild Card' at \$299.99 USD/YEAR (ESTIMATED). Both SKUs offer 1 Year, X.509 v3, RSA-SHA256, Auto Renew, and Improve SEO. The W1 SKU also includes a Wild Card feature. A note at the bottom of the left window states: 'Create certificate operation may take 1-10 minutes to complete. Once created, App Service Certificates can only be used by other App Services within the same subscription.'

Use the following table to help you configure the certificate. When finished, click **Create**.

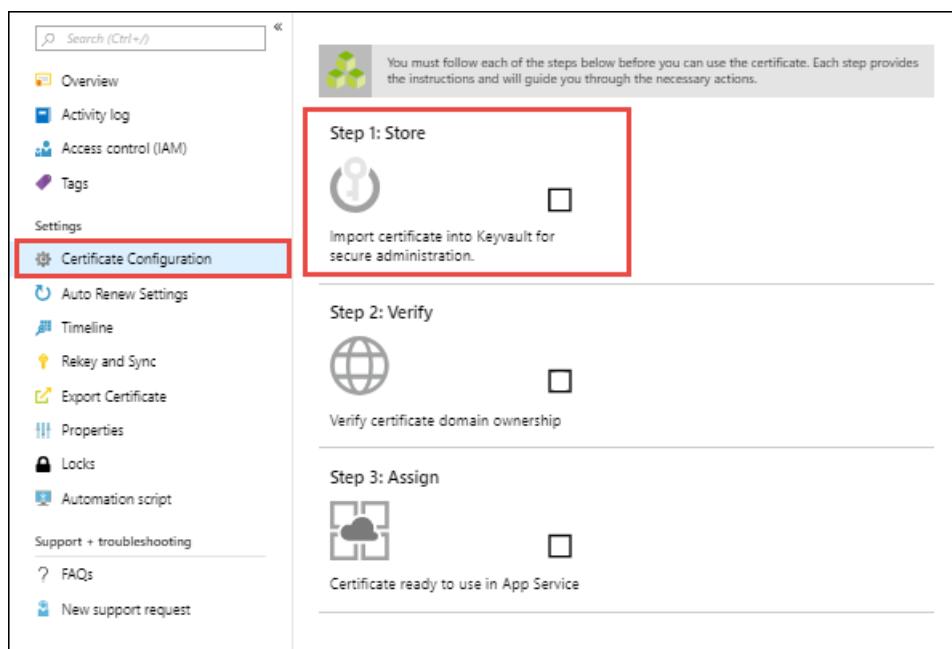
SETTING	DESCRIPTION
Name	A friendly name for your App Service certificate.
Naked Domain Host Name	Specify the root domain here. The issued certificate secures <i>both</i> the root domain and the <code>www</code> subdomain. In the issued certificate, the Common Name field contains the root domain, and 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.

SETTING	DESCRIPTION
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.

### 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**.



[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> .

SETTING	DESCRIPTION
Access policies	Defines the applications and the allowed access to the vault resources. You can configure it later, following the steps at <a href="#">Grant several applications access to a key vault</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.

### Verify domain ownership

From the same **Certificate Configuration** page you used in the last step, click **Step 2: Verify**.

The screenshot shows the Azure Key Vault Certificate Configuration page. On the left, there's a sidebar with various options like Overview, Activity log, Access control (IAM), Tags, Settings, Certificate Configuration (which is selected and highlighted in blue), Auto Renew Settings, Timeline, Rekey and Sync, Export Certificate, Properties, Locks, Automation script, Support + troubleshooting, FAQs, and New support request. The main content area has three steps: Step 1: Store (green checkmark, certificate successfully imported), Step 2: Verify (red box highlights this step, with a note to verify certificate domain ownership), and Step 3: Assign (greyed out, certificate ready to use in App Service).

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.

### 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**.

The screenshot shows the Azure portal interface for managing app service bindings. The left sidebar has a 'TLS/SSL settings' section with several options, one of which is 'Import App Service Certificate', highlighted with a red box. The main content area shows the 'Private Key Certificates (.pfx)' tab selected under 'Bindings'. It includes instructions for using .pfx certificates for TLS/SSL, four action buttons ('Import App Service Certificate', 'Upload Certificate', 'Import Key Vault Certificate', 'Create App Service Managed Certificate'), and a table for listing private key certificates. The table has columns for Health St..., Hostname, Expiration, and Thumbprint, and a status filter at the top.

Select the certificate that you just purchased and select **OK**.

When the operation completes, you see the certificate in the **Private Key Certificates** list.

The screenshot shows the 'Private Key Certificates' list. It has a 'Status Filter' at the top with 'All' selected. Below is a table with columns for Health Status, Hostname, Expiration, and Thumbprint. One row is shown, indicating a 'Healthy' status for the hostname 'contoso.com, www.contoso.com' with an expiration date of '10/10/2020' and a thumbprint starting with '6A3BCCA5CC4B0158F0A097CE9F39...'. There is also a '...' button for more actions.

#### 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](#).

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**.

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**

Status Filter: All, Healthy, Warning, Expired

Health St...	Hostname	Expiration	Thumbprint
No private key certificates available for app.			

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](#).

Health Status	Hostname	Expiration	Thumbprint	More
<span style="color: green;">✓</span> 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](#).

## 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 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 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**.

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. In the main content area, under the Bindings section, the 'Private Key Certificates (.pfx)' tab is selected. It displays instructions for using PFX files for TLS/SSL bindings. Below this, there are four buttons: 'Import App Service Certificate', 'Upload Certificate' (which is also highlighted with a red box), 'Import Key Vault Certificate', and 'Create App Service Managed Certificate'. A table titled 'Private Key Certificates' shows a single entry: a green checkmark icon, the status 'Healthy', the hostname 'www.contoso.com', the expiration date '4/11/2020', and the thumbprint '6A3BCCA5CC4B0158F0A097CE9F39...'. There are also columns for Health St., Hostname, Expiration, and Thumbprint.

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.

This screenshot shows the 'Private Key Certificates' list after a certificate has been uploaded. The table has columns for Health Status, Hostname, Expiration, and Thumbprint. One row is visible, showing a green checkmark icon, the status 'Healthy', the hostname 'www.contoso.com', the expiration date '4/11/2020', and the thumbprint '6A3BCCA5CC4B0158F0A097CE9F39...'. There is also a '...' button next to the thumbprint.

### 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)** X

 **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  ✓

\* CER Certificate file  📁

**Upload**

Once the certificate is uploaded, copy the certificate thumbprint and see [Make the certificate accessible](#).

## Manage App Service certificates

This section shows you how to manage an App Service certificate you purchased in [Import an App Service certificate](#).

- [Rekey certificate](#)
- [Renew certificate](#)
- [Export certificate](#)
- [Delete 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.

Search (Ctrl+ /) Refresh Rekey Sync

Overview Activity log Access control (IAM) Tags

Settings Certificate Configuration Auto Renew Settings Timeline Rekey and Sync Export Certificate Properties Locks Automation script

Support + troubleshooting FAQs New support request

## Rekey Certificate

Rekeying your certificate will roll the certificate with a new certificate issued from the certificate authority. While Rekeying your certificate will go through Pending Issuance state and once the certificate is ready make sure you sync your resources using this certificate to prevent disruption to service.

Certificate rekey operations are free and rekey does not incur additional charges.

### Linked Private Certificate

Linked Private Certificates are used in App Service apps. The private certificates can go out of sync after rekey and renew operation. Click sync to update the private certificates and the SSL bindings to the latest certificate.

Current Certificate Thumbprint	ID		
STATUS	LINKED PRIVATE CERTIFI...	RESOURCE GROUP	THUMBPRINT
Healthy	myResourceGro...		

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 48 hours.

## Renew certificate

To turn on automatic renewal of your 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** and click **Save**. Certificates can start automatically renewing 60 days before expiration if you have automatic renewal turned on.

Search (Ctrl+ /) Refresh Save Discard Manual Renew Sync

Overview Activity log Access control (IAM) Tags

Settings Certificate Configuration Auto Renew Settings Timeline Rekey and Sync Export Certificate Properties Locks Automation script

Support + troubleshooting FAQs New support request

### Manual renewal not allowed at this time

App Service Certificate is not eligible for manual renewal right now. Manual renewal will become available 60 days before expiry to prevent accidental renewal. Use Manual Renew ONLY if you really need to get a new certificate issued with extended expiry. For rolling keys use the Rekey feature. Turn off the auto renew feature to opt out of automatic auto renewal.

Auto Renew App Service Certificate

### Linked Private Certificate

Linked Private Certificates are used in App Service apps. The private certificates can go out of sync after rekey and renew operation. Click sync to update the private certificates and the SSL bindings to the latest certificate.

Current Certificate Thumbprint	ID		
STATUS	LINKED PRIVATE CERTIFI...	RESOURCE GROUP	THUMBPRINT
Healthy	myResourceGro...		

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 48 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. 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
contoso	Delete	contoso	Deleting this App Service cert...

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 an SSL binding](#)
- [Enforce HTTPS](#)
- [Enforce TLS 1.1/1.2](#)
- [Use an SSL certificate in your application code](#)
- [FAQ : App Service Certificates](#)

# Configure your App Service app to use Azure AD login

2/28/2020 • 5 minutes to read • [Edit Online](#)

This article shows you how to configure Azure App Service to use Azure Active Directory (Azure AD) as an authentication provider.

Follow these best practices when setting up your app and authentication:

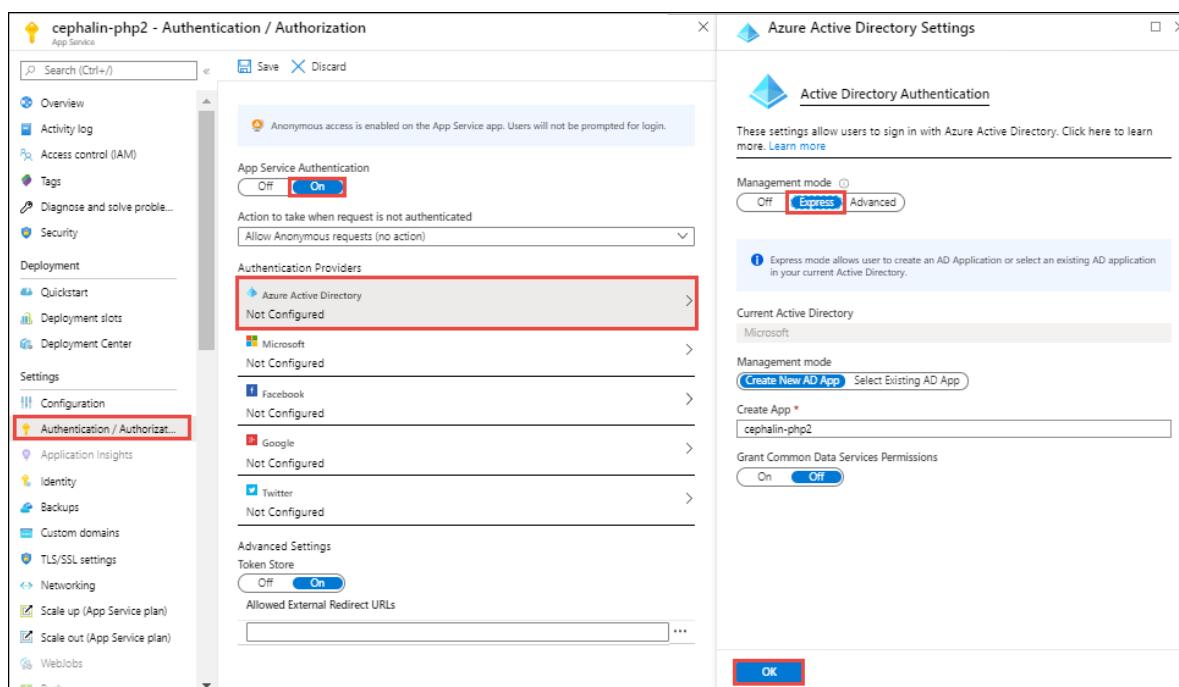
- Give each App Service app its own permissions and consent.
- Configure each App Service app with its own registration.
- Avoid permission sharing between environments by using separate app registrations for separate deployment slots. When testing new code, this practice can help prevent issues from affecting the production app.

## Configure with express settings

### NOTE

The **Express** option is not available for government clouds.

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
  2. From the left navigation, select **Authentication / Authorization > On**.
  3. Select **Azure Active Directory > Express**.
- If you want to choose an existing app registration instead:
- a. Choose **Select Existing AD app**, then click **Azure AD App**.
  - b. Choose an existing app registration and click **OK**.
4. Select **OK** to register the App Service app in Azure Active Directory. A new app registration is created.



5. (Optional) By default, App Service provides authentication but doesn't restrict authorized access to your site content and APIs. You must authorize users in your app code. To restrict app access only to users authenticated by Azure Active Directory, set **Action to take when request is not authenticated** to **Log in with Azure Active Directory**. When you set this functionality, your app requires all requests to be authenticated. It also redirects all unauthenticated to Azure Active Directory for authentication.

**Caution**

Restricting access in this way applies to all calls to your app, which might not be desirable for apps that have a publicly available home page, as in many single-page applications. For such applications, **Allow anonymous requests (no action)** might be preferred, with the app manually starting login itself. For more information, see [Authentication flow](#).

6. Select **Save**.

## Configure with advanced settings

You can configure app settings manually if you want to use an app registration from a different Azure AD tenant. To complete this custom configuration:

1. Create a registration in Azure AD.
2. Provide some of the registration details to App Service.

### Create an app registration in Azure AD for your App Service app

You'll need the following information when you configure your App Service app:

- Client ID
- Tenant ID
- Client secret (optional)
- Application ID URI

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. Select **Azure Active Directory > App registrations > 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 **Create**.
6. After the app registration is created, copy the **Application (client) ID** and the **Directory (tenant) ID** for later.
7. Select **Branding**. In **Home page URL**, enter the URL of your App Service app and select **Save**.
8. Select **Expose an API > Set**. Paste in the URL of your App Service app and select **Save**.

**NOTE**

This value is the **Application ID URI** of the app registration. If your web app requires access to an API in the cloud, you need the **Application ID URI** of the web app when you configure the cloud App Service resource. You can use this, for example, if you want the cloud service to explicitly grant access to the web app.

9. Select **Add a scope**.
  - a. In **Scope name**, enter *user\_impersonation*.
  - b. In the text boxes, enter the consent scope name and description you want users to see on the consent page. For example, enter *Access my app*.
  - c. Select **Add scope**.

10. (Optional) To create a client secret, select **Certificates & secrets > New client secret > Add**. Copy the client secret value shown in the page. It won't be shown again.

11. (Optional) To add multiple **Reply URLs**, select **Authentication**.

#### Enable Azure Active Directory in your App Service app

1. In the [Azure portal](#), search for and select **App Services**, and then select your app.
2. In the left pane, under **Settings**, select **Authentication / Authorization > On**.
3. (Optional) By default, App Service authentication allows unauthenticated access to your app. To enforce user authentication, set **Action to take when request is not authenticated** to **Log in with Azure Active Directory**.
4. Under **Authentication Providers**, select **Azure Active Directory**.
5. In **Management mode**, select **Advanced** and configure App Service authentication according to the following table:

FIELD	DESCRIPTION
Client ID	Use the <b>Application (client) ID</b> of the app registration.
Issuer ID	Use <code>https://login.microsoftonline.com/&lt;tenant-id&gt;</code> , and replace <code>&lt;tenant-id&gt;</code> with the <b>Directory (tenant) ID</b> of the app registration.
Client Secret (Optional)	Use the client secret you generated in the app registration.
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.

6. Select **OK**, and then select **Save**.

You're now ready to use Azure Active Directory for authentication in your App Service app.

## Configure a native client application

You can register native clients to allow authentication using a client library such as the **Active Directory Authentication Library**.

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 Windows 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. Select **user\_impersonation**, and then select **Add permissions**.

You have now configured a native client application that can access your App Service app.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Advanced usage of authentication and authorization in Azure App Service](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#).

# Configure your App Service app to use Facebook login

12/2/2019 • 2 minutes to read • [Edit Online](#)

This article shows how to configure Azure App Service 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

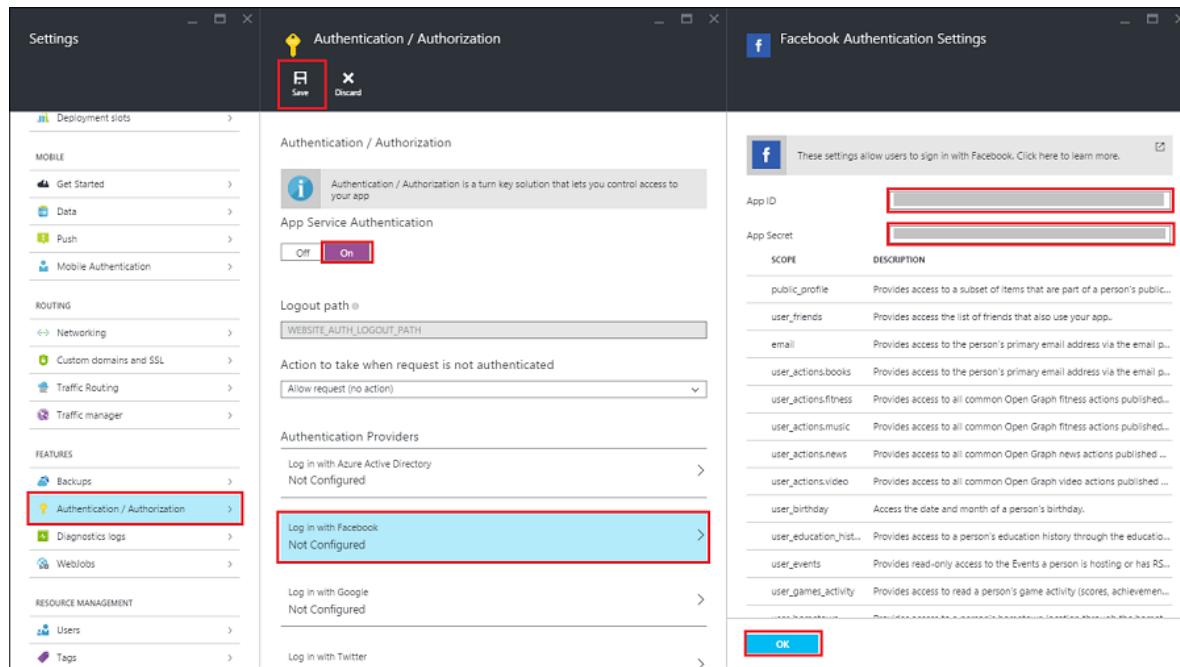
The app secret is an important security credential. Do not share this secret with anyone or distribute it within a client application.

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 Service app.
2. Select **Settings > Authentication / Authorization**, and make sure that **App Service Authentication** is **On**.
3. Select **Facebook**, and then paste in the App ID and App Secret values that you obtained previously. Enable any scopes needed by your application.
4. Select **OK**.



By default, App Service provides authentication, but it doesn't restrict authorized access to your site content and APIs. You need to authorize users in your app code.

5. (Optional) To restrict access only to users authenticated by Facebook, set **Action to take when request is not authenticated** to **Facebook**. When you set this functionality, your app requires all requests to be authenticated. It also redirects all unauthenticated requests to Facebook for authentication.

#### Caution

Restricting access in this way applies to all calls to your app, which might not be desirable for apps that have a publicly available home page, as in many single-page applications. For such applications, **Allow anonymous requests (no action)** might be preferred so that the app manually starts authentication itself. For more information, see [Authentication flow](#).

6. Select **Save**.

You're now ready to use Facebook for authentication in your app.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Advanced usage of authentication and authorization in Azure App Service](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#).

# Configure your App Service app to use Google login

12/2/2019 • 2 minutes to read • [Edit Online](#)

This topic shows you how to configure Azure App Service 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](https://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. In the [Azure portal](#), go to your App Service app.
2. Select **Settings > Authentication / Authorization**, and make sure that **App Service Authentication** is **On**.
3. Select **Google**, then paste in the App ID and App Secret values that you obtained previously. Enable any scopes needed by your application.
4. Select **OK**.

App Service provides authentication but doesn't restrict authorized access to your site content and APIs. For more information, see [Authorize or deny users](#).

5. (Optional) To restrict site access only to users authenticated by Google, set **Action to take when request is not authenticated** to **Google**. When you set this functionality, your app requires that all requests be authenticated. It also redirects all unauthenticated requests to Google for authentication.

### Caution

Restricting access in this way applies to all calls to your app, which might not be desirable for apps that have a publicly available home page, as in many single-page applications. For such applications, **Allow anonymous requests (no action)** might be preferred so that the app manually starts authentication itself. For more information, see [Authentication flow](#).

6. Select **Save**.

You are now ready to use Google for authentication in your app.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Advanced usage of authentication and authorization in Azure App Service](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#).

# Configure your App Service app to use Microsoft Account login

1/28/2020 • 2 minutes to read • [Edit Online](#)

This topic shows you how to configure Azure App Service to use AAD to support personal Microsoft account logins.

## NOTE

Both personal Microsoft accounts and organizational accounts use the AAD identity provider. At this time, it is not possible to configure this identity provider to support both types of log-ins.

## Register your app with Microsoft Account

1. Go to [App registrations](#) in the Azure portal. If needed, sign in with your Microsoft account.
2. Select **New registration**, then enter an application name.
3. Under **Supported account types**, select **Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)**
4. In **Redirect URIs**, select **Web**, and then enter `https://<app-domain-name>/.auth/login/aad/callback`. Replace `<app-domain-name>` with the domain name of your app. For example, `https://contoso.azurewebsites.net/.auth/login/aad/callback`. Be sure to use the HTTPS scheme in the URL.
5. Select **Register**.
6. Copy the **Application (Client) ID**. You'll need it later.
7. From the left pane, select **Certificates & secrets > New client secret**. Enter a description, select the validity duration, and select **Add**.
8. Copy the value that appears on the **Certificates & secrets** page. After you leave the page, it won't be displayed again.

## IMPORTANT

The client secret value (password) is an important security credential. Do not share the password with anyone or distribute it within a client application.

## Add Microsoft Account information to your App Service application

1. Go to your application in the [Azure portal](#).
2. Select **Settings > Authentication / Authorization**, and make sure that **App Service Authentication** is **On**.
3. Under **Authentication Providers**, select **Azure Active Directory**. Select **Advanced** under **Management mode**. Paste in the Application (client) ID and client secret that you obtained earlier. Use <https://login.microsoftonline.com/9188040d-6c67-4c5b-b112-36a304b66dad/v2.0> for the **Issuer**

**Url** field.

4. Select **OK**.

App Service provides authentication, but doesn't restrict authorized access to your site content and APIs. You must authorize users in your app code.

5. (Optional) To restrict access to Microsoft account users, set **Action to take when request is not authenticated** to **Log in with Azure Active Directory**. When you set this functionality, your app requires all requests to be authenticated. It also redirects all unauthenticated requests to use AAD for authentication. Note that because you have configured your **Issuer Url** to use the Microsoft Account tenant, only personal accounts will successfully authenticate.

**Caution**

Restricting access in this way applies to all calls to your app, which might not be desirable for apps that have a publicly available home page, as in many single-page applications. For such applications, **Allow anonymous requests (no action)** might be preferred so that the app manually starts authentication itself. For more information, see [Authentication flow](#).

6. Select **Save**.

You are now ready to use Microsoft Account for authentication in your app.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Advanced usage of authentication and authorization in Azure App Service](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#).

# Configure your App Service app to use Twitter login

12/2/2019 • 2 minutes to read • [Edit Online](#)

This article shows how to configure Azure App Service 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 New App**.
3. Enter a **Name** and a **Description** for your new app. Paste your application's **URL** into the **Website** field. In the **Callback URL** field, enter the URL of your App Service app and append the path `/auth/login/aad/callback`. For example, `https://contoso.azurewebsites.net/.auth/login/twitter/callback`. Make sure to use the HTTPS scheme.
4. At the bottom of the page, read and accept the terms. Select **Create your Twitter application**. The application details are displayed.
5. Select the **Settings** tab, check **Allow this application to be used to sign in with Twitter**, and then select **Update Settings**.
6. Select the **Keys and Access Tokens** tab.

Make a note of these values:

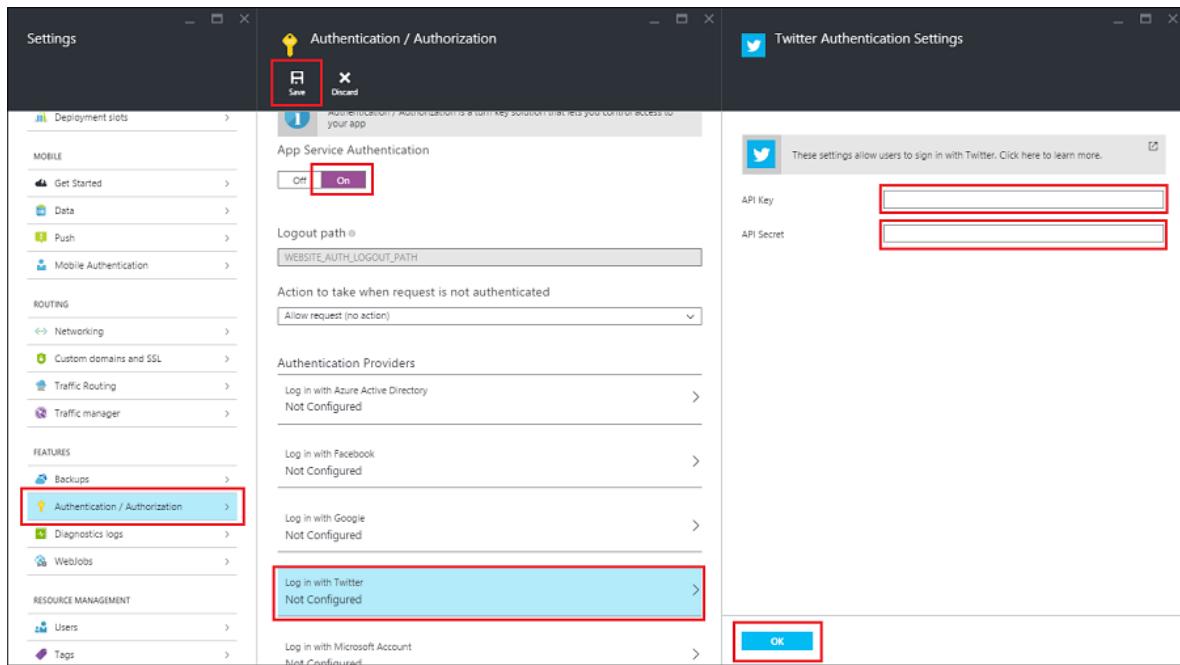
- Consumer key (API key)
- Consumer secret (API secret)

### NOTE

The consumer secret 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. Go to your application in the [Azure portal](#).
2. Select **Settings > Authentication / Authorization**, and make sure that **App Service Authentication** is **On**.
3. Select **Twitter**.
4. Paste in the `API Key` and `API Secret` values that you obtained previously.
5. Select **OK**.



By default, App Service provides authentication but doesn't restrict authorized access to your site content and APIs. You must authorize users in your app code.

6. (Optional) To restrict access to your site to only users authenticated by Twitter, set **Action to take when request is not authenticated** to **Twitter**. When you set this functionality, your app requires all requests to be authenticated. It also redirects all unauthenticated requests to Twitter for authentication.

#### Caution

Restricting access in this way applies to all calls to your app, which might not be desirable for apps that have a publicly available home page, as in many single-page applications. For such applications, **Allow anonymous requests (no action)** might be preferred so that the app manually starts authentication itself. For more information, see [Authentication flow](#).

7. Select **Save**.

You are now ready to use Twitter for authentication in your app.

## Next steps

- [App Service Authentication / Authorization overview](#).
- [Advanced usage of authentication and authorization in Azure App Service](#)
- Add authentication to your Mobile App: [iOS](#), [Android](#), [Windows Universal](#), [Xamarin.Android](#), [Xamarin.iOS](#), [Xamarin.Forms](#), [Cordova](#).

# Advanced usage of authentication and authorization in Azure App Service

12/2/2019 • 10 minutes to read • [Edit Online](#)

This article shows you how to customize the built-in [authentication and authorization in App Service](#), and to manage identity from your application.

To get started quickly, see one of the following tutorials:

- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service \(Windows\)](#)
- [Tutorial: Authenticate and authorize users end-to-end in Azure App Service for Linux](#)
- [How to configure your app to use Azure Active Directory login](#)
- [How to configure your app to use Facebook login](#)
- [How to configure your app to use Google login](#)
- [How to configure your app to use Microsoft Account login](#)
- [How to configure your app to use Twitter login](#)

## 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:

```
<a href="/.auth/login/aad">Log in with Azure AD</a>
<a href="/.auth/login/microsoftaccount">Log in with Microsoft Account</a>
<a href="/.auth/login/facebook">Log in with Facebook</a>
<a href="/.auth/login/google">Log in with Google</a>
<a href="/.auth/login/twitter">Log in with Twitter</a>
```

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_url` 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_url=/Home/Index">Log in</a>
```

## Validate tokens from providers

In a client-directed sign-in, the application signs in the user to the provider manually and then submits the authentication token to App Service for validation (see [Authentication 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>"}	
microsoftaccount	{"access_token": "<token>"}	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. When specified, it can also optionally be accompanied by the <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:

```
<a href="/.auth/logout">Sign out</a>
```

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"
```

## Access user claims

App Service passes user claims to your application by using special 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. For ASP.NET 4.6 apps, the **ClaimsPrincipal** is automatically set with the appropriate values. ASP.NET Core, however, doesn't provide an authentication middleware that integrates with App Service user claims. For a workaround, see [MaximeRouiller.Azure.AppService.EasyAuth](#).

Your application 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](#).

## 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
Microsoft Account	X-MS-TOKEN-MICROSOFTACCOUNT-ACCESS-TOKEN X-MS-TOKEN-MICROSOFTACCOUNT-EXPIRES-ON X-MS-TOKEN-MICROSOFTACCOUNT-AUTHENTICATION-TOKEN X-MS-TOKEN-MICROSOFTACCOUNT-REFRESH-TOKEN
Twitter	X-MS-TOKEN-TWITTER-ACCESS-TOKEN X-MS-TOKEN-TWITTER-ACCESS-TOKEN-SECRET

From your client code (such as a mobile app or in-browser JavaScript), send an HTTP `GET` request to `/auth/me`. 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 identity provider 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 Account:** When [configuring Microsoft Account Authentication Settings](#), select the `wl.offline_access` scope.
- **Azure Active Directory:** 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** > **authsettings**.
  3. Click **Edit**.
  4. Modify the following property. Replace `<app_id>` with the Azure Active Directory application ID of the service you want to access.

```
"additionalLoginParams": ["response_type=code id_token", "resource=<app_id>"]
```

5. Click **Put**.

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.

## 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](#).

## Next steps

[Tutorial: Authenticate and authorize users end-to-end \(Windows\)](#) [Tutorial: Authenticate and authorize users end-to-end \(Linux\)](#)

# Configure TLS mutual authentication for Azure App Service

1/6/2020 • 6 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.

## Enable client certificates

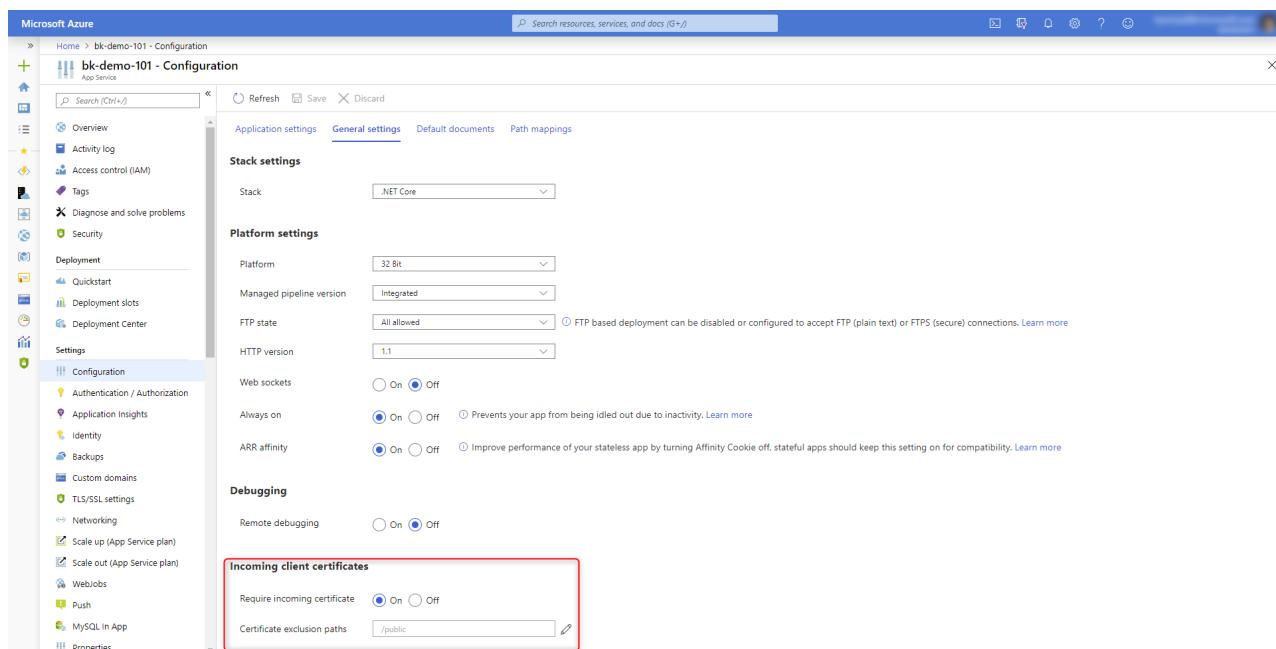
To set up your app to require client certificates, you need to set the `clientCertEnabled` setting for your app to `true`. To set the setting, run the following command in the [Cloud Shell](#).

```
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 will require a client certificate for access. To allow certain paths to remain open for anonymous access, you can define exclusion paths as part of your application configuration.

Exclusion paths can be configured by selecting **Configuration > General Settings** and defining an exclusion path. In this example, anything under `/public` path for your application would not request a client certificate.



## Access client certificate

In App Service, SSL 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 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;
            }
        }
    }
}

// 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 thumbprint 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[] mdBase64 = this.getCertificate().getEncoded();
        byte[] thumbprintBase64 = DatatypeConverter.parseBase64Binary(this.getThumbprint());
        return md.digest(mdBase64).equals(thumbprintBase64);
    }
}
```

```
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;
}
}
```

# Scale up an app in Azure App Service

1/3/2020 • 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 **PremiumV2** tier, see [Configure PremiumV2 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, Scale up (App Service plan) (highlighted with a red box), Scale out (App Service plan), WebJobs, Push, MySQL In App, Properties, Locks, and Export template. Under 'App Service plan', it shows 'App Service plan' (selected), 'Quotas', and 'Change App Service plan'. At the bottom of the sidebar is a 'Development Tools' section.

The main area displays three pricing tiers: F1 (Dev / Test), D1 (Production), and B1 (Isolated). Each tier has a description, memory, compute, and cost. A red box highlights the D1 tier. Below the tiers is a 'See additional options' link. To the right, sections for 'Included features' (Custom domains) and 'Included hardware' (Azure Compute Units (ACU), Memory, Storage) are shown. An 'Apply' button is at the bottom.

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 'Overview' (highlighted with a red box), Activity log, Access control (IAM), Tags, Diagnose and solve problems, Security, and Deployment. The main area shows deployment details: 'Resource group (change) myResourceGroup', 'Status Running', 'Location West Europe', 'Subscription (change) mySubscription', and 'Subscription ID 00000000-0000-0000-000000000000'. A purple bar above the details says 'Click here to access our Quickstart guide for deploying code to your app →'. To the right, deployment information is listed: URL (<https://my-demo-app>), App Service Plan ([myAppServicePlan](#)), FTP/deployment user ([my-demo-app\user](#)), FTP hostname (<ftp://waws-prod-am>), and FTPS hostname (<ftps://waws-prod-am>).

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.

NAME	TYPE
dotnetcoredb	SQL server
coreDB (dotnetcoredb/coreDB)	SQL database
myAppServicePlan	App Service plan
my-demo-app	App Service

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 PremiumV2 tier for App Service](#)

2 minutes to read

# High-density hosting on Azure App Service using per-app scaling

12/2/2019 • 3 minutes to read • [Edit Online](#)

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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** 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.

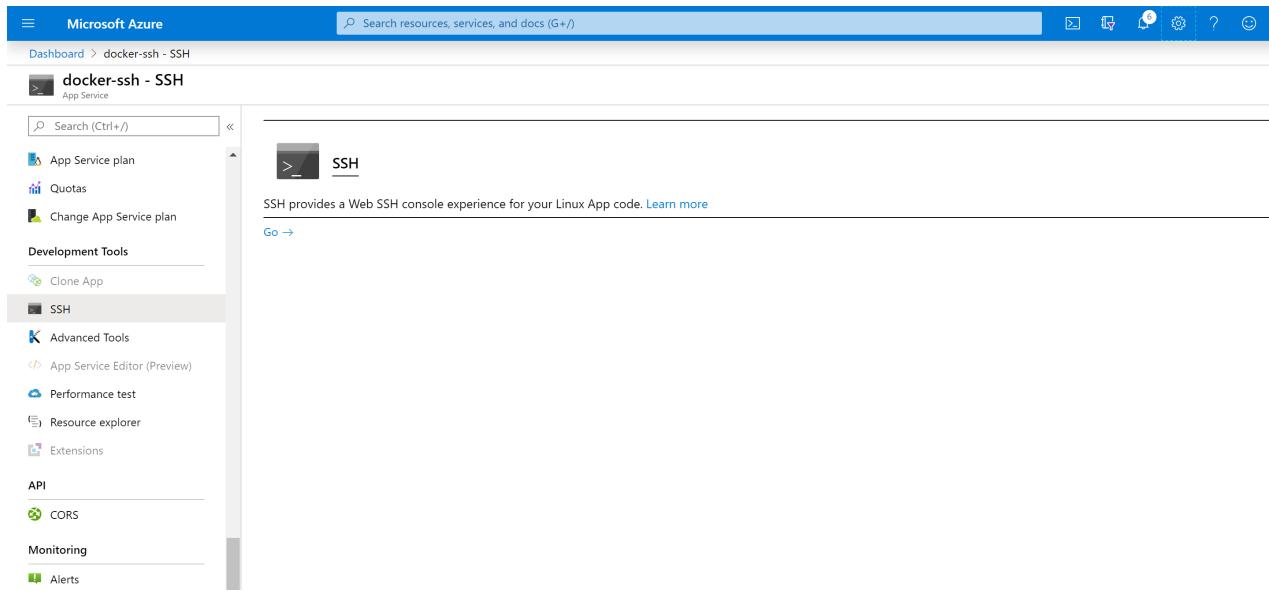
## Next steps

- [Azure App Service plans in-depth overview](#)
- [Introduction to App Service Environment](#)

# SSH support for Azure App Service on Linux

1/30/2020 • 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 navigation bar with 'Microsoft Azure' and a search bar. Below the navigation bar, the URL 'Dashboard > docker-ssh - SSH' is visible. The main content area is titled 'docker-ssh - SSH' under 'App Service'. On the left, there's a sidebar with various options like 'App Service plan', 'Quotas', 'Change App Service plan', 'Development Tools' (which has 'SSH' selected), 'Clone App', 'Advanced Tools', 'App Service Editor (Preview)', 'Performance test', 'Resource explorer', 'Extensions', 'API', 'CORS', 'Monitoring', and 'Alerts'. The main panel shows a large button labeled 'SSH' with the sub-instruction 'SSH provides a Web SSH console experience for your Linux App code. Learn more'. Below this, there's a 'Go →' button.

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
[ - ]  rmmnologin
[ - ]  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 remote-connection create` 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.

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
0e690efafa93e2:~#
```

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](#)
- [How to use a custom Docker image for Web App for Containers](#)
- [Using .NET Core in Azure App Service on Linux](#)
- [Using Ruby in Azure App Service on Linux](#)
- [Azure App Service Web App for Containers FAQ](#)

# Monitor apps in Azure App Service

1/3/2020 • 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* that are 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
<b>CPU (Short)</b>	The amount of CPU allowed for this app in a 5-minute interval. This quota resets every five minutes.
<b>CPU (Day)</b>	The total amount of CPU allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
<b>Memory</b>	The total amount of memory allowed for this app.
<b>Bandwidth</b>	The total amount of outgoing bandwidth allowed for this app in a day. This quota resets every 24 hours at midnight UTC.
<b>Filesystem</b>	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 restarted.

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 you have been whitelisted for private preview.

Metrics provide information about the app or the App Service plan's behavior.

For an app, the available metrics are:

METRIC	DESCRIPTION
<b>Average Response Time</b>	The average time taken for the app to serve requests, in seconds.
<b>Average memory working set</b>	The average amount of memory used by the app, in megabytes (MiB).
<b>Connections</b>	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().
<b>CPU Time</b>	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> .
<b>Current Assemblies</b>	The current number of Assemblies loaded across all AppDomains in this application.

METRIC	DESCRIPTION
<b>Data In</b>	The amount of incoming bandwidth consumed by the app, in MiB.
<b>Data Out</b>	The amount of outgoing bandwidth consumed by the app, in MiB.
<b>File System Usage</b>	Percentage of filesystem quota consumed by the app.
<b>Gen 0 Garbage Collections</b>	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.
<b>Gen 1 Garbage Collections</b>	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.
<b>Gen 2 Garbage Collections</b>	The number of times the generation 2 objects are garbage collected since the start of the app process.
<b>Handle Count</b>	The total number of handles currently open by the app process.
<b>Http 2xx</b>	The count of requests resulting in an HTTP status code $\geq 200$ but $< 300$ .
<b>Http 3xx</b>	The count of requests resulting in an HTTP status code $\geq 300$ but $< 400$ .
<b>Http 401</b>	The count of requests resulting in HTTP 401 status code.
<b>Http 403</b>	The count of requests resulting in HTTP 403 status code.
<b>Http 404</b>	The count of requests resulting in HTTP 404 status code.
<b>Http 406</b>	The count of requests resulting in HTTP 406 status code.
<b>Http 4xx</b>	The count of requests resulting in an HTTP status code $\geq 400$ but $< 500$ .
<b>Http Server Errors</b>	The count of requests resulting in an HTTP status code $\geq 500$ but $< 600$ .
<b>IO Other Bytes Per Second</b>	The rate at which the app process is issuing bytes to I/O operations that don't involve data, such as control operations.
<b>IO Other Operations Per Second</b>	The rate at which the app process is issuing I/O operations that aren't read or write operations.
<b>IO Read Bytes Per Second</b>	The rate at which the app process is reading bytes from I/O operations.
<b>IO Read Operations Per Second</b>	The rate at which the app process is issuing read I/O operations.

METRIC	DESCRIPTION
<b>IO Write Bytes Per Second</b>	The rate at which the app process is writing bytes to I/O operations.
<b>IO Write Operations Per Second</b>	The rate at which the app process is issuing write I/O operations.
<b>Memory working set</b>	The current amount of memory used by the app, in MiB.
<b>Private Bytes</b>	Private Bytes is the current size, in bytes, of memory that the app process has allocated that can't be shared with other processes.
<b>Requests</b>	The total number of requests regardless of their resulting HTTP status code.
<b>Requests In Application Queue</b>	The number of requests in the application request queue.
<b>Thread Count</b>	The number of threads currently active in the app process.
<b>Total App Domains</b>	The current number of AppDomains loaded in this application.
<b>Total App Domains Unloaded</b>	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
<b>CPU Percentage</b>	The average CPU used across all instances of the plan.
<b>Memory Percentage</b>	The average memory used across all instances of the plan.
<b>Data In</b>	The average incoming bandwidth used across all instances of the plan.
<b>Data Out</b>	The average outgoing bandwidth used across all instances of the plan.
<b>Disk Queue Length</b>	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.
<b>Http Queue Length</b>	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.

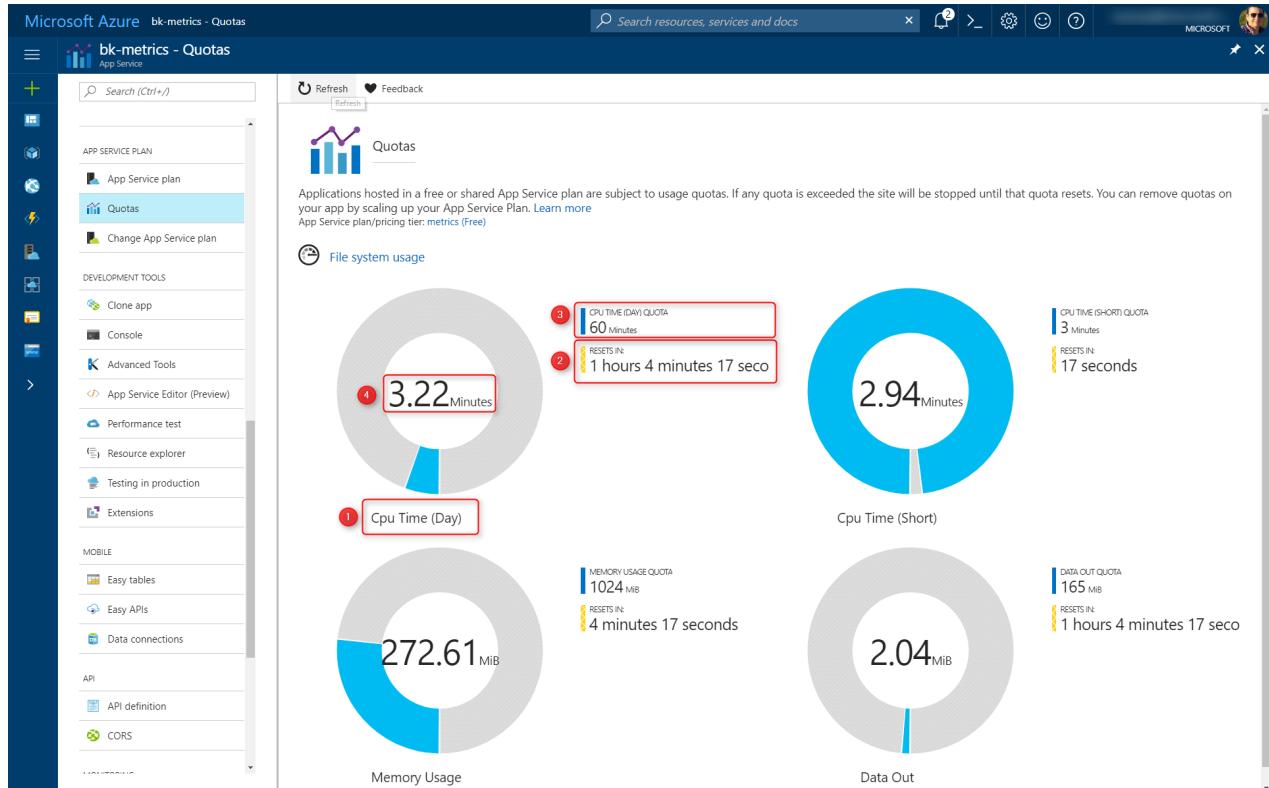
## Metrics granularity and retention policy

Metrics for an app and app service plan are logged and aggregated by the service, with the following granularities and retention policies:

- **Minute** granularity metrics are kept for 30 hours.
- **Hour** granularity metrics are kept for 30 days.
- **Day** granularity metrics are kept for 30 days.

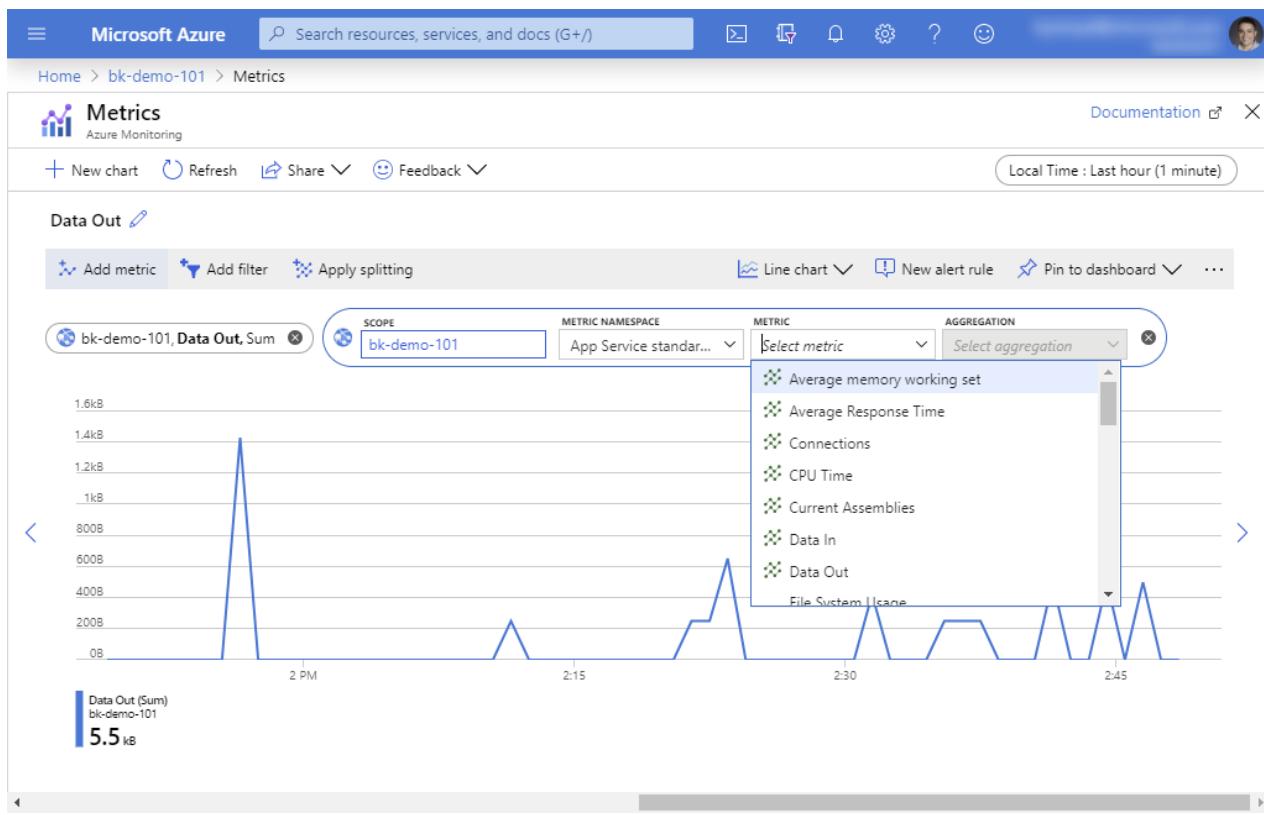
## Monitoring quotas and metrics in the Azure portal

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

2/4/2020 • 8 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 \(preview\)](#) 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 .htm 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)

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
<b>Disabled</b>	None
<b>Error</b>	Error, Critical
<b>Warning</b>	Warning, Error, Critical
<b>Information</b>	Info, Warning, Error, Critical
<b>Verbose</b>	Trace, Debug, Info, Warning, Error, Critical (all categories)

When finished, select **Save**.

## 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**.

## 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 exceed 50, the oldest 26 errors are automatically deleted.

## 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](#).

## 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:

```
az webapp log tail --name appname --resource-group myResourceGroup
```

To filter specific events, such as errors, use the **--Filter** parameter. For example:

```
az webapp log tail --name appname --resource-group myResourceGroup --filter Error
```

To filter specific log types, such as HTTP, use the **--Path** parameter. For example:

```
az webapp log tail --name appname --resource-group myResourceGroup --path 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
<b>Application logs</b>	<code>/LogFiles/Application/</code>	Contains one or more text files. The format of the log messages depends on the logging provider you use.
<b>Failed Request Traces</b>	<code>/LogFiles/W3SVC#####/</code>	Contains XML files, and an XSL file. You can view the formatted XML files in the browser.
<b>Detailed Error Logs</b>	<code>/LogFiles/DetailedErrors/</code>	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.
<b>Web Server Logs</b>	<code>/LogFiles/http/RawLogs/</code>	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.

LOG TYPE	DIRECTORY	DESCRIPTION
<b>Deployment logs</b>	/LogFiles/Git/ and /deployments/	Contain logs generated by the internal deployment processes, as well as logs for Git deployments.

## Send logs to Azure Monitor (preview)

With the new [Azure Monitor integration](#), you can [create Diagnostic Settings \(preview\)](#) to send logs to Storage Accounts, Event Hubs and Log Analytics.

The screenshot shows the Azure portal interface for managing diagnostic settings. On the left, there's a sidebar with various tools like 'Clone App', 'SSH', 'Advanced Tools', 'App Service Editor (Preview)', 'Performance test', 'Resource explorer', 'Extensions', 'API' (with 'CORS', 'Monitoring', 'Alerts', 'Metrics'), and 'Diagnostic settings'. The 'Diagnostic settings' item is highlighted with a red box. The main content area shows 'Subscription \*' set to 'Demo Two Subscription' and 'Resource group' set to 'appsvc-azmon-rg'. Below this, it says 'Demo Two Subscription > appsvc-azmon-rg > appsvc-azmon-app'. Under 'Diagnostics settings', there's a table with columns 'Name', 'Storage account', and 'Event hub'. A row shows 'No diagnostic settings defined' and a button '+ Add diagnostic setting' which is also highlighted with a red box. Below the table, a note says 'Click 'Add Diagnostic setting' above to configure the collection of the following data:' followed by a list of items: AppServiceHTTPLogs, AppServiceConsoleLogs, AppServiceAppLogs, AppServiceFileAuditLogs, AppServiceAuditLogs, and AllMetrics.

### Supported log types

The following table shows the supported log types and descriptions:

LOG TYPE	WINDOWS SUPPORT	LINUX (DOCKER) SUPPORT	DESCRIPTION
AppServiceConsoleLogs	TBA	Yes	Standard output and standard error
AppServiceHTTPLogs	Yes	Yes	Web server logs
AppServiceEnvironmentPlatformLogs	Yes	Yes	App Service Environment: scaling, configuration changes, and status logs
AppServiceAuditLogs	Yes	Yes	Login activity via FTP and Kudu

LOG TYPE	WINDOWS SUPPORT	LINUX (DOCKER) SUPPORT	DESCRIPTION
AppServiceFileAuditLogs	TBA	Yes	File changes via FTP and Kudu
AppServiceAppLogs	TBA	Java SE & Tomcat	Application logs

## 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](#)

# Back up your app in Azure

12/2/2019 • 6 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** tier or **Premium** tier. For more information about scaling your App Service plan to use a higher tier, see [Scale up an app in Azure](#).  
**Premium** tier allows 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. If the backup size exceeds this limit, you get an error.
- Backups of SSL enabled Azure Database for MySQL is not supported. If a backup is configured, you will get failed backups.
- Backups of SSL enabled Azure Database for PostgreSQL is not supported. If a backup is configured, you will get failed backups.
- In-app MySQL databases are automatically backed up without any configuration. If you make manually 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 get failed backups.

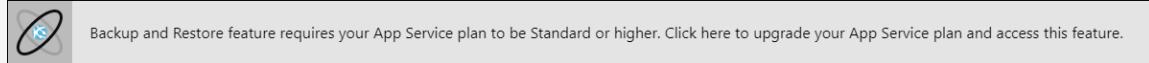
## Create a manual backup

1. In the [Azure portal](#), navigate to your app's page, select **Backups**. The **Backups** page is displayed.

The screenshot shows the 'SETTINGS' section of the Azure App Service configuration. The 'Backups' option is highlighted with a red box. Other visible options include Application settings, Authentication / Authorization, Custom domains, SSL certificates, Networking, and Scale up (App Service plan).

#### 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 this, a message reads: 'Configure backup to create restorable archive copies of your apps content, configuration and database.' A link 'Learn more' is provided. A red box highlights the message 'Backup is not configured. Click here to configure backup for your app.' which is preceded by a gear icon.

3. In the **Backup Configuration** page, click **Storage not configured** to configure a storage account.

The screenshot shows the 'Backup Storage' configuration page. It features a grey server icon and the word 'Backup Storage'. Below this, a message says: 'Select the target container to store your app backup.' A red box highlights the link 'Storage not configured' under the heading 'Storage Settings'.

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**.



## Backup Database

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.



INCLUDE IN BACKUP

CONNECTION STRING NAME

DATABASE TYPE

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 manually settings for in-app MySQL databases, 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**.



## Backup

Configure backup to create restorable archive copies of your apps content, configuration and database. [Learn more](#)



Backup configured, backup schedule is not configured, configure scheduled backup to automatically take backups.



Backup



Restore

STATUS

BACKUP TIME

SIZE (MB)



InProgress

Wednesday, October 16, 2019, 6:20:10 PM

0

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.

## 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 blacklisted directories, or any blacklisted 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 a SQL database, this is a BACPAC file (no file extension) and can be imported. To create a SQL database based on the BACPAC export, see [Import a BACPAC File to Create a New User Database](#).

### WARNING

Altering any of the files in your **websitebackups** container can cause the backup to become invalid and therefore non-restorable.

## 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

12/2/2019 • 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 the 'contoso-backup' app service. The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, and Deployment options. The 'Backups' option is highlighted with a red box. The main content area has a 'Backup' section with a warning icon and text about configuring backups. Below it are 'Configure', 'Backup', and 'Restore' buttons. The 'Restore' button is also highlighted with a red box. A table at the bottom shows a single backup entry with a checkmark, the date 'Thursday, June 8, 2017..', and size '0.06 MB'.

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 a collapsible arrow. The essentials include:

- Resource group: cephalin-appwithsql
- Status: Primary: Available
- Location: West Europe
- Subscription name: Visual Studio Ultimate with MSDN
- Subscription ID: [REDACTED]

On the right side of the essentials section are four icons: a key, a lightning bolt, two people, and a tag.

Below the essentials is a 'Performance/Access tier' section with a pencil icon, showing Standard/Hot. It also lists Replication (Locally-redundant storage (LRS)) and the Blob service endpoint (<https://cephalinstorage4.blob.core.windows..>).

At the bottom of the essentials section is a link to 'All settings →'.

Below the essentials is a search bar labeled 'Search containers by prefix' with a magnifying glass icon.

The main content area displays a table of 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	...

4. Select backup file you want to download or delete.

The screenshot shows the Azure Storage Explorer interface. At the top, there's a toolbar with icons for Refresh, Delete container, Properties, and Access policy. Below the toolbar is a search bar with the placeholder text "Search blobs by prefix (case-sensitive)". The main area displays a table of blobs in the "backups" container. The columns are 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](#)

# Azure App Service App Cloning Using PowerShell

1/14/2020 • 5 minutes to read • [Edit Online](#)

## NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

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 manager 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
```

## 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

## 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](#)

# Move resources to a new resource group or subscription

1/10/2020 • 9 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 a SQL Server and its database to a new resource group, an application that uses the database experiences no downtime. It can still read and write to the database. 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.

## 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've moving any of the following services, check that guidance before moving.
  - [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](#)
3. 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.
4. 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](#)

5. 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
```

6. The account moving the resources must have at least the following permissions:

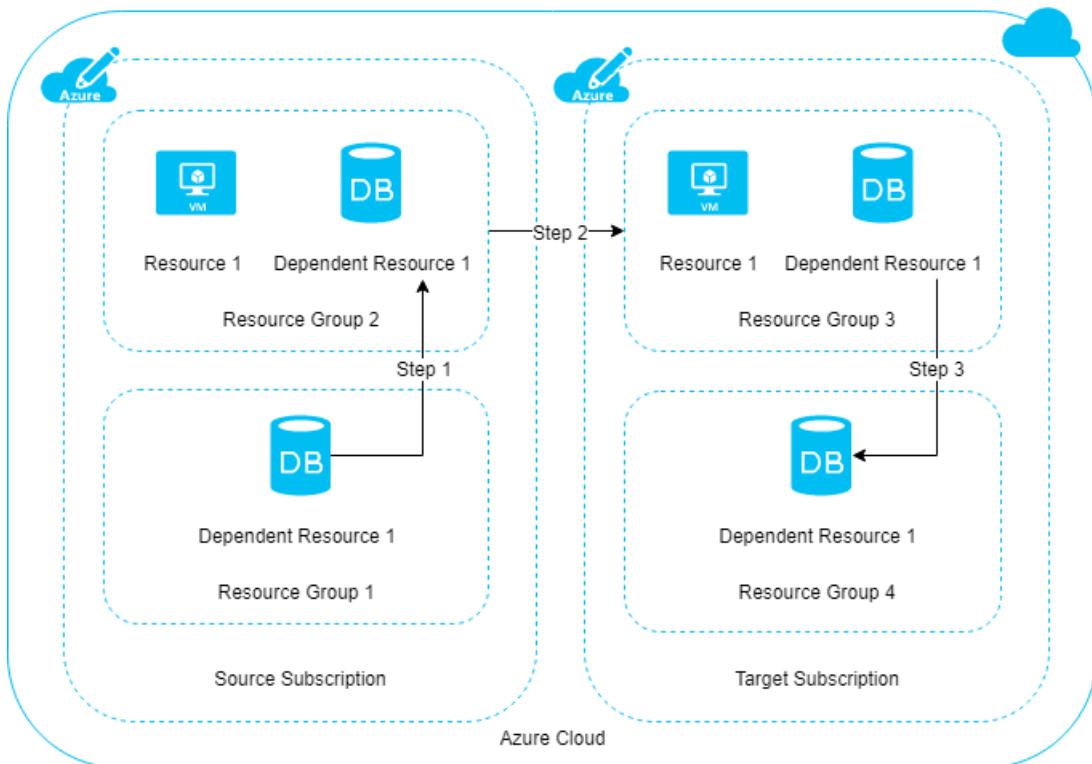
- **Microsoft.Resources/subscriptions/resourceGroups/moveResources/action** on the source resource group.
  - **Microsoft.Resources/subscriptions/resourceGroups/write** on the destination resource group.
7. Before moving the resources, check the subscription quotas for the subscription you're moving 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](#).
8. **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 consolidated 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.

## Validate move

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:

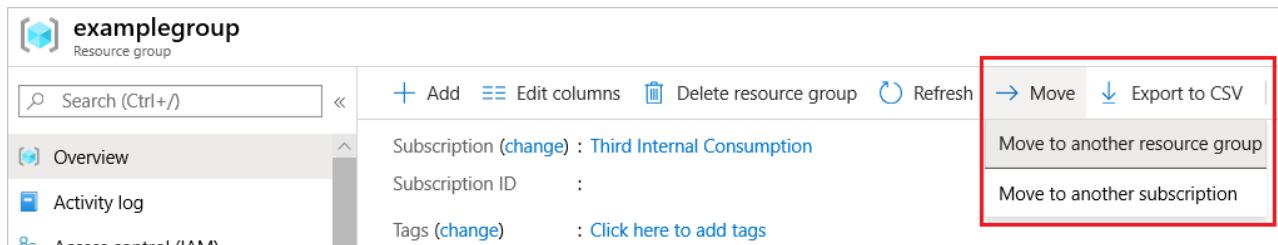
```
GET <location-url>  
Authorization: Bearer <access-token>
```

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>..."}}
```

## Use the portal

To move resources, select the resource group with those resources, and then select the **Move** button.



The screenshot shows the Azure portal interface for managing a resource group named 'examplegroup'. On the left, there's a navigation sidebar with links for Overview, Activity log, and Access control (IAM). The main area displays resource group details: Subscription (change) : Third Internal Consumption, Subscription ID : [redacted], and Tags (change) : Click here to add tags. In the top right corner, there's a toolbar with buttons for Add, Edit columns, Delete resource group, Refresh, Move, and Export to CSV. The 'Move' button is highlighted with a red box. A dropdown menu from the 'Move' button lists two options: 'Move to another resource group' and 'Move to another subscription'.

Select whether you're moving the resources to a new resource group or a new subscription.

Select the resources to move and the destination resource group. Acknowledge that you need to update scripts for these resources and select **OK**. If you selected the edit subscription icon in the previous step, you must also select the destination subscription.

## Move resources

### Resources to move

<input checked="" type="checkbox"/> Select all	Type
<input checked="" type="checkbox"/>	Failure Anomalies - tfexamplesite
<input checked="" type="checkbox"/>	Application Insights Smart Detection
<input checked="" type="checkbox"/>	tfexamplesite
<input checked="" type="checkbox"/>	ASP-examplegroup-a26c
<input checked="" type="checkbox"/>	tfexamplesite

### Move these resources to

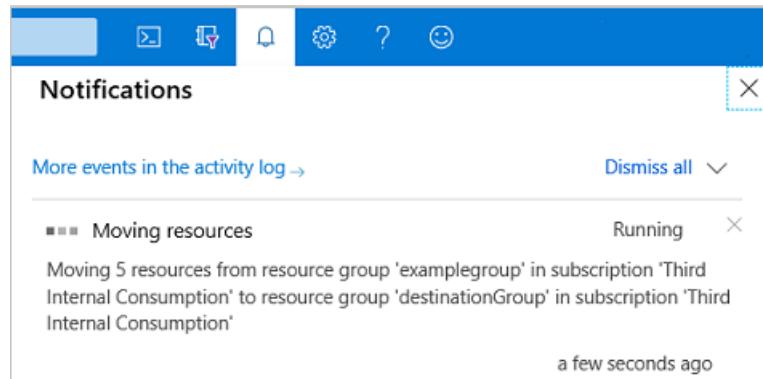
#### Resource group \*

destinationGroup

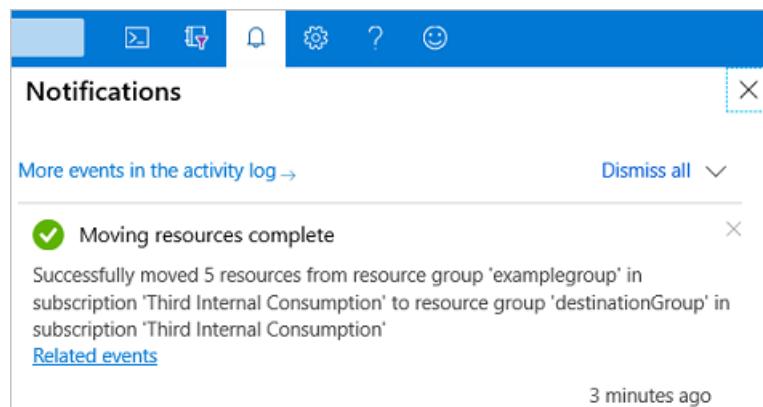
I understand that tools and scripts associated with moved resources will not work until I update them to use new resource IDs [\(i\)](#)

**OK**

In **Notifications**, you see that the move operation is running.



When it has completed, you're notified of the result.



If you get an error, see [Troubleshoot moving Azure resources to new resource group or subscription](#).

## Use Azure PowerShell

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.

If you get an error, see [Troubleshoot moving Azure resources to new resource group or subscription](#).

## Use Azure CLI

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.

If you get an error, see [Troubleshoot moving Azure resources to new resource group or subscription](#).

## Use REST API

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>"
}
```

If you get an error, see [Troubleshoot moving Azure resources to new resource group or subscription](#).

## 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 4-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 4 hours during resource move?

The 4-hour window is the maximum time allowed for resource move. To prevent modifications on the resources being moved, both the source and destination resource groups are locked for the duration of 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 4-hour window when a resource provider fails either phase. During the allowed time, Resource Manager retries the failed step.

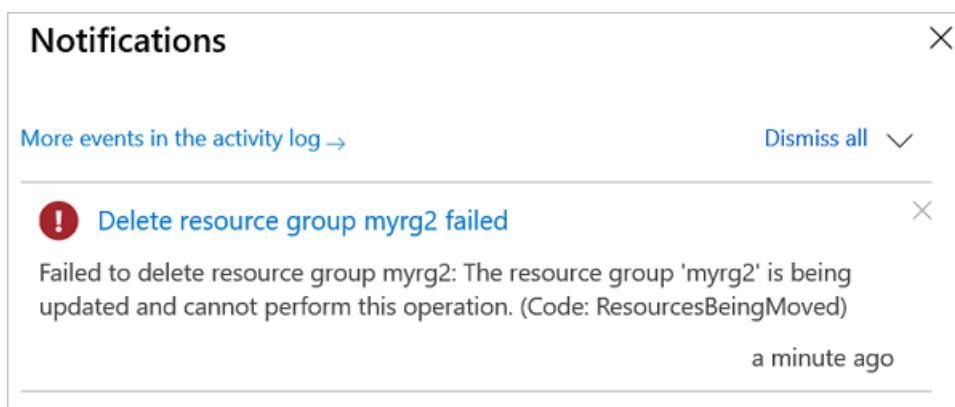
If a resource can't be moved within the 4-hour window, Resource Manager unlocks both resource groups.

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: 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](#).

## Next steps

For a list of which resources support move, see [Move operation support for resources](#).

# Azure App Service on Linux FAQ

2/18/2020 • 6 minutes to read • [Edit Online](#)

With the release of App Service on Linux, we're working on adding features and making improvements to our platform. This article provides answers to questions that our customers have been asking us recently.

If you have a question, comment on this article.

## Built-in images

### I want to fork the built-in Docker containers that the platform provides. Where can I find those files?

You can find all Docker files on [GitHub](#). You can find all Docker containers on [Docker Hub](#).

### What are the expected values for the Startup File section when I configure the runtime stack?

STACK	EXPECTED VALUE
Java SE	the command to start your JAR app (for example, <code>java -jar /home/site/wwwroot/app.jar --server.port=80</code> )
Tomcat	the location of a script to perform any necessary configurations (for example, <code>/home/site/deployments/tools/startup_script.sh</code> )
Node.js	the PM2 configuration file or your script file
.Net Core	the compiled DLL name as <code>dotnet &lt;myapp&gt;.dll</code>
Ruby	the Ruby script that you want to initialize your app with

These commands or scripts are executed after the built-in Docker container is started, but before your application code is started.

## Management

### What happens when I press the restart button in the Azure portal?

This action is the same as a Docker restart.

### Can I use Secure Shell (SSH) to connect to the app container virtual machine (VM)?

Yes, you can do that through the source control management (SCM) site.

#### NOTE

You can also connect to the app container directly from your local development machine using SSH, SFTP, or Visual Studio Code (for live debugging Node.js apps). For more information, see [Remote debugging and SSH in App Service on Linux](#).

### How can I create a Linux App Service plan through an SDK or an Azure Resource Manager template?

You should set the **reserved** field of the app service to *true*.

## Continuous integration and deployment

**My web app still uses an old Docker container image after I've updated the image on Docker Hub. Do you support continuous integration and deployment of custom containers?**

Yes, to set up continuous integration/deployment for Azure Container Registry or DockerHub, by following [Continuous Deployment with Web App for Containers](#). For private registries, you can refresh the container by stopping and then starting your web app. Or you can change or add a dummy application setting to force a refresh of your container.

**Do you support staging environments?**

Yes.

**Can I use *WebDeploy/MSDeploy* to deploy my web app?**

Yes, you need to set an app setting called `WEBSITE_WEBDEPLOY_USE_SCM` to *false*.

**Git deployment of my application fails when using Linux web app. How can I work around the issue?**

If Git deployment fails to your Linux web app, choose one of the following options to deploy your application code:

- Use the Continuous Delivery (Preview) feature: You can store your app's source code in an Azure DevOps Git repo or GitHub repo to use Azure Continuous Delivery. For more information, see [How to configure Continuous Delivery for Linux web app](#).
- Use the [ZIP deploy API](#): To use this API, [SSH into your web app](#) and go to the folder where you want to deploy your code. Run the following code:

```
curl -X POST -u <user> --data-binary @<zipfile> https://<your-sitename>.scm.azurewebsites.net/api/zipdeploy
```

If you get an error that the `curl` command is not found, make sure you install curl by using `apt-get install curl` before you run the previous `curl` command.

## Language support

**I want to use web sockets in my Node.js application, any special settings, or configurations to set?**

Yes, disable `perMessageDeflate` in your server-side Node.js code. For example, if you are using socket.io, use the following code:

```
const io = require('socket.io')(server,{  
  perMessageDeflate :false  
});
```

**Do you support uncompiled .NET Core apps?**

Yes.

**Do you support Composer as a dependency manager for PHP apps?**

Yes, during a Git deployment, Kudu should detect that you're deploying a PHP application (thanks to the presence of a composer.lock file), and Kudu will then trigger a composer install.

## Custom containers

**I'm using my own custom container. I want the platform to mount an SMB share to the `/home/` directory.**

If `WEBSITES_ENABLE_APP_SERVICE_STORAGE` setting is **unspecified** or set to *true*, the `/home/` directory **will be shared** across scale instances, and files written **will persist** across restarts. Explicitly setting `WEBSITES_ENABLE_APP_SERVICE_STORAGE` to *false* will disable the mount.

**My custom container takes a long time to start, and the platform restarts the container before it finishes starting up.**

You can configure the amount of time the platform will wait before it restarts your container. To do so, set the `WEBSITES_CONTAINER_START_TIME_LIMIT` app setting to the value you want. The default value is 230 seconds, and the maximum value is 1800 seconds.

**What is the format for the private registry server URL?**

Provide the full registry URL, including `http://` or `https://`.

**What is the format for the image name in the private registry option?**

Add the full image name, including the private registry URL (for example, `myacr.azurecr.io/dotnet:latest`). Image names that use a custom port [cannot be entered through the portal](#). To set `docker-custom-image-name`, use the `az` [command-line tool](#).

**Can I expose more than one port on my custom container image?**

We don't support exposing more than one port.

**Can I bring my own storage?**

Yes, [bring your own storage](#) is in preview.

**Why can't I browse my custom container's file system or running processes from the SCM site?**

The SCM site runs in a separate container. You can't check the file system or running processes of the app container.

**My custom container listens to a port other than port 80. How can I configure my app to route requests to that port?**

We have automatic port detection. You can also specify an app setting called `WEBSITES_PORT` and give it the value of the expected port number. Previously, the platform used the `PORT` app setting. We are planning to deprecate this app setting and to use `WEBSITES_PORT` exclusively.

**Do I need to implement HTTPS in my custom container?**

No, the platform handles HTTPS termination at the shared front ends.

## Multi-container with Docker Compose

**How do I configure Azure Container Registry (ACR) to use with multi-container?**

In order to use ACR with multi-container, **all container images** need to be hosted on the same ACR registry server. Once they are on the same registry server, you will need to create application settings and then update the Docker Compose configuration file to include the ACR image name.

Create the following application settings:

- `DOCKER_REGISTRY_SERVER_USERNAME`

- DOCKER\_REGISTRY\_SERVER\_URL (full URL, ex: `https://<server-name>.azurecr.io`)
- DOCKER\_REGISTRY\_SERVER\_PASSWORD (enable admin access in ACR settings)

Within the configuration file, reference your ACR image like the following example:

```
image: <server-name>.azurecr.io/<image-name>:<tag>
```

### **How do I know which container is internet accessible?**

- Only one container can be open for access
- Only port 80 and 8080 is accessible (exposed ports)

Here are the rules for determining which container is accessible - in the order of precedence:

- Application setting `WEBSITES_WEB_CONTAINER_NAME` set to the container name
- The first container to define port 80 or 8080
- If neither of the above is true, the first container defined in the file will be accessible (exposed)

## Pricing and SLA

### **What is the pricing, now that the service is generally available?**

You are charged the normal Azure App Service pricing for the number of hours that your app runs.

## Other questions

### **What are the supported characters in application settings names?**

You can use only letters (A-Z, a-z), numbers (0-9), and the underscore character (\_) for application settings.

### **Where can I request new features?**

You can submit your idea at the [Web Apps feedback forum](#). Add "[Linux]" to the title of your idea.

## Next steps

- [What is Azure App Service on Linux?](#)
- [Set up staging environments in Azure App Service](#)
- [Continuous Deployment with Web App for Containers](#)

# Azure subscription and service limits, quotas, and constraints

2/25/2020 • 85 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

If you want to raise the limit or quota above the default limit, [open an online customer support request at no charge](#). The limits can't be raised above the maximum limit value shown in the following tables. If there's no maximum limit column, the resource doesn't have adjustable limits.

[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 make a specific request for Azure resource group 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 Azure resource group 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](#).

### Subscription limits

The following limits apply when you use Azure Resource Manager and Azure resource groups.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Subscriptions per Azure Active Directory tenant	Unlimited.	Unlimited.
Coadministrators per subscription	Unlimited.	Unlimited.
Resource groups per subscription	980	980

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure Resource Manager API request size	4,194,304 bytes.	4,194,304 bytes.
Tags per subscription <sup>1</sup>	Unlimited.	Unlimited.
Unique tag calculations per subscription <sup>1</sup>	10,000	10,000
<a href="#">Subscription-level deployments</a> per location	800 <sup>2</sup>	800

<sup>1</sup>You can apply an unlimited number of tags per 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 10,000 or less. You still can find a resource by tag when the number exceeds 10,000.

<sup>2</sup>If you reach the limit of 800 deployments, delete deployments from the history that are no longer needed. To delete subscription level deployments, use [Remove-AzDeployment](#) or [az deployment delete](#).

## Resource group limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Resources per <a href="#">resource group</a>	N/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>	800
Resources per deployment	800	800
Management locks per unique scope	20	20
Number of tags per resource or resource group	50	50
Tag key length	512	512
Tag value length	256	256

<sup>1</sup>If you reach the limit of 800 deployments per resource group, delete deployments from the history that are no longer needed. Deleting an entry from the deployment history doesn't affect the deployed resources. For more information, see [Resolve error when deployment count exceeds 800](#).

## Template limits

VALUE	DEFAULT LIMIT	MAXIMUM LIMIT
Parameters	256	256

Value	Default Limit	Maximum Limit
Variables	256	256
Resources (including copy count)	800	800
Outputs	64	64
Template expression	24,576 chars	24,576 chars
Resources in exported templates	200	200
Template size	4 MB	4 MB
Parameter file size	64 KB	64 KB

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](#).

## Active Directory limits

Here are the usage constraints and other service limits for the Azure Active Directory (Azure AD) service.

Category	Limits
Directories	A single user can belong to a maximum of 500 Azure AD directories as a member or a guest. A single user can create a maximum of 20 directories.
Domains	You can add no more than 900 managed domain names. If you set up all of your domains for federation with on-premises Active Directory, you can add no more than 450 domain names in each directory.
Resources	<ul style="list-style-type: none"> <li>A maximum of 50,000 Azure AD resources can be created in a single directory by users of the Free edition of Azure Active Directory by default. If you have at least one verified domain, the default directory service quota in Azure AD is extended to 300,000 Azure AD resources.</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>

CATEGORY	LIMITS
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> <li>Schema extensions are available only in the Graph API version 1.21 preview. The application must be granted write access to register an extension.</li> </ul>
Applications	A maximum of 100 users can be owners of a single application.
Application Manifest	A maximum of 1200 entries can be added in the Application Manifest.

CATEGORY	LIMITS
Groups	<ul style="list-style-type: none"> <li>A user can create a maximum of 250 groups in an Azure AD organization.</li> <li>An Azure AD organization can have a maximum of 5000 dynamic groups.</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.</li> <li>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.</li> <li>Nested Groups in Azure AD are not supported within all scenarios</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 the signed-in user is a member of are included)</li> <li>Conditional access (when scoping a conditional access policy to a group)</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> <li>Office 365 Groups.</li> </ul>
Application Proxy	<ul style="list-style-type: none"> <li>A maximum of 500 transactions per second per App Proxy application</li> <li>A maximum of 750 transactions per second for the Azure AD organization</li> </ul> <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>

CATEGORY	LIMITS
Access Panel	<ul style="list-style-type: none"> <li>There's no limit to the number of applications that can be seen in the Access Panel per user. This applies to users assigned licenses for Azure AD Premium or the Enterprise Mobility Suite.</li> <li>A maximum of 10 app tiles can be seen in the Access Panel for each user. This limit applies to users who are assigned licenses for Azure AD Free license plan. Examples of app tiles include Box, Salesforce, or Dropbox. This limit doesn't apply to administrator accounts.</li> </ul>
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.
Admin roles and permissions	<ul style="list-style-type: none"> <li>A group cannot be added as an <a href="#">owner</a>.</li> <li>A group cannot be assigned to a <a href="#">role</a>.</li> <li>Users' ability to read other users' directory information cannot be restricted outside of the Azure AD organization-wide switch to disable all non-admin users' access to all directory information (not recommended). More information on default permissions <a href="#">here</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	10 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
Maximum number of CA certificates per service instance	10
Maximum number of service instances per subscription <sup>7</sup>	20
Maximum number of subscriptions per service instance <sup>7</sup>	500
Maximum number of client certificates per service instance <sup>7</sup>	50

RESOURCE	LIMIT
Maximum number of APIs per service instance <sup>7</sup>	50
Maximum number of API operations per service instance <sup>7</sup>	1,000
Maximum total request duration <sup>7</sup>	30 seconds
Maximum buffered payload size <sup>7</sup>	2 MiB
Maximum request URL size <sup>8</sup>	4096 bytes

<sup>1</sup>Scaling limits depend on the pricing tier. To see 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 4 KiB.

<sup>6</sup>This resource is available in the Premium tier only.

<sup>7</sup>This resource applies to the Consumption tier only.

<sup>8</sup>Applies to the Consumption tier only. Includes an up to 2048 bytes long query string.

## App Service limits

The following App Service limits include limits for Web Apps, Mobile Apps, and API Apps.

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V2)	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>	30 dedicated <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>
CPU time (5 minutes) <sup>6</sup>	3 minutes	3 minutes	Unlimited, pay at standard rates			

Resource	Free	Shared	Basic	Standard	Premium (V2)	Isolated
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
IP connections	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
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 per plan			5	25	200	200
Integrated load balancer		X	X	X	X	X <sup>10</sup>
Always On			X	X	X	X

RESOURCE	FREE	SHARED	BASIC	STANDARD	PREMIUM (V2)	ISOLATED
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
WebJobs <sup>11</sup>	X	X	X	X	X	X
Endpoint monitoring			X	X	X	X
Staging slots				5	20	20
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 500GB.

<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/P1V2 instance, 3,968 per B2/S2/P2V2 instance, 8,064 per B3/S3/P3V2 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.

## Automation limits

### Process automation

Resource	Maximum 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	1MB	A single stream cannot be larger than 1 MB.
Maximum number of modules that can be imported every 30 seconds per Automation account	5	
Maximum size of a module	100 MB	
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.
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 Hybrid Worker Groups 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 kilobits	
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.

RESOURCE	MAXIMUM LIMIT	NOTES
Maximum webhook payload size	512 kilobits	
Maximum days that job data is retained	30 days	
Maximum PowerShell workflow state size	5 MB	Applies to PowerShell workflow runbooks when checkpointing workflow.

<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	
Registry	250	
Windows software	250	Doesn't include software updates.
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	

## 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	DEFAULT LIMIT	MAXIMUM LIMIT
Web or worker roles per deployment <sup>1</sup>	25	25
Instance input endpoints per deployment	25	25
Input endpoints per deployment	25	25
Internal endpoints per deployment	25	25
Hosted service certificates per deployment	199	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

Resource	Free	Basic	S1	S2	S3	S3 HD	L1	L2
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

Storage is constrained by disk space or by a hard limit on the *maximum number* of indexes, document, or other high-level resources, whichever comes first. The following table documents storage limits. For maximum limits on indexes, documents, and other objects, see [Limits by resource](#).

Resource	Free	Basic <sup>1</sup>	S1	S2	S3	S3 HD <sup>2</sup>	L1	L2
Service level agreement (SLA) <sup>3</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
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. At this tier, additional search units are used for allocating more replicas for increased query workloads.

<sup>2</sup> S3 HD has a hard limit of three partitions, which is lower than the partition limit for S3. The lower partition limit is imposed because the index count for S3 HD is substantially higher. Given that service limits exist for both computing resources (storage and processing) and content (indexes and documents), the content limit is reached first.

<sup>3</sup> Service level agreements are offered 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 additional limitations, for more information see [Azure Cognitive Services](#).

Type	Limit	Example
A mixture of Cognitive Services resources	Maximum of 200 total Cognitive Services resources.	100 Computer Vision resources in West US 2, 50 Speech Service resources in West US, and 50 Text Analytics resources in East US.
A single type of Cognitive Services resources.	Maximum of 100 resources per region, with a maximum of 200 total Cognitive Services resources.	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 attached database configurations in a cluster	70

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	App Service Plan <sup>1</sup>
Scale out	Event driven	Event driven	Manual/autoscale

RESOURCE	CONSUMPTION PLAN	PREMIUM PLAN	APP SERVICE PLAN
Max instances	200	100	10-20
Default <a href="#">timeout duration</a> (min)	5	30	30 <sup>2</sup>
Max <a href="#">timeout duration</a> (min)	10	unbounded <sup>8</sup>	unbounded <sup>3</sup>
Max outbound connections (per instance)	600 active (1200 total)	unbounded	unbounded
Max request size (MB) <sup>4</sup>	100	100	100
Max query string length <sup>4</sup>	4096	4096	4096
Max request URL length <sup>4</sup>	8192	8192	8192
<a href="#">ACU</a> per instance	100	210-840	100-840
Max memory (GB per instance)	1.5	3.5-14	1.75-14
Function apps per plan	100	100	unbounded <sup>5</sup>
<a href="#">App Service plans</a>	100 per <a href="#">region</a>	100 per resource group	100 per resource group
Storage <sup>6</sup>	1 GB	250 GB	50-1000 GB
Custom domains per app	500 <sup>7</sup>	500	500
Custom domain <a href="#">SSL support</a>	unbounded SNI SSL connection included	unbounded SNI SSL and 1 IP SSL connections included	unbounded SNI SSL and 1 IP SSL connections included

<sup>1</sup> For specific limits for the various App Service plan options, see the [App Service plan limits](#).

<sup>2</sup> By default, the timeout for the Functions 1.x runtime in an App Service plan is unbounded.

<sup>3</sup> Requires the App Service plan be set to [Always On](#). Pay at standard [rates](#).

<sup>4</sup> These limits are [set in the host](#).

<sup>5</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>6</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>7</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>8</sup> Guaranteed for up to 60 minutes.

## Azure Kubernetes Service limits

RESOURCE	DEFAULT LIMIT
Maximum clusters per subscription	100

RESOURCE	DEFAULT LIMIT
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 (100 nodes per <a href="#">node pool</a> )
Maximum pods per node: <a href="#">Basic networking</a> with Kubenet	110
Maximum pods per node: <a href="#">Advanced networking</a> with Azure Container Networking Interface	Azure CLI deployment: 30 <sup>1</sup> Azure Resource Manager template: 30 <sup>1</sup> Portal deployment: 30

<sup>1</sup>When you deploy an Azure Kubernetes Service (AKS) cluster with the Azure CLI or a Resource Manager template, this value is configurable up to 250 pods per node. You can't configure maximum pods per node after you've already deployed an AKS cluster, or if you deploy a cluster by using the Azure portal.

## 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 data size limit for Azure Maps. The Azure Maps data service is available only at the S1 pricing tier.

RESOURCE	LIMIT
Maximum size of data	50 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	1000 active alert rules per subscription in Azure public, Azure China 21Vianet and Azure Government clouds.	Call support.
Activity log alerts	100 active alert rules per subscription.	Same as default.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Log alerts	512	Call support.
Action groups	2,000 action groups per subscription.	Call support.
Autoscale settings	100 per region per subscription.	Same as default.

## Action groups

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Azure app push	10 Azure app actions per action group.	Call support.
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> .	Call support.
ITSM	10 ITSM actions in an action group.	Call support.
Logic app	10 logic app actions in an action group.	Call support.
Runbook	10 runbook actions in an action group.	Call support.
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> .	Call support.
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> .	Call support.
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> .	Call support.

## Log queries and language

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.

LIMIT	DESCRIPTION
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.
Query throttling	A user is limited to 200 queries per 30 seconds on any number of workspaces. This limit applies to programmatic queries or to queries initiated by visualization parts such as Azure dashboards and the Log Analytics workspace summary page.

## 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.
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	Limits	Comments
Maximum records returned by a log query	10,000	Reduce results using query scope, time range, and filters in the query.

## Data Collector API

Category	Limits	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.

## Search API

Category	Limits	Comments
Maximum records returned in a single query	500,000	
Maximum size of data returned	64,000,000 bytes (~61 MiB)	
Maximum query running time	10 minutes	See <a href="#">Timeouts</a> for details.
Maximum request rate	200 requests per 30 seconds per AAD user or client IP address	See <a href="#">Rate limits</a> for details.

## General workspace limits

Category	Limits	Comments
Maximum columns in a table	500	
Maximum characters for column name	500	
Data export	Not currently available	Use Azure Function or Logic App to aggregate and export data.

## 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 default ingestion volume rate limit for data sent from Azure resources using [diagnostic settings](#) is approximately **6 GB/min** per workspace. This is an approximate value since the actual size can vary between data types depending on the log length and its compression ratio. This limit does not apply to

data that is sent from agents or [Data Collector API](#).

If you send data at a higher rate to a single workspace, 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 continues to exceed the rate limit or you are expecting to reach it sometime soon, you can request an increase to your workspace by opening a support request.

To be notified on such an event in your workspace, create a [log alert rule](#) using the following query with alert logic base on number of results grater than zero.

```
Operation  
|where OperationCategory == "Ingestion"  
|where Detail startswith "The rate of data crossed the threshold"
```

#### 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	90 days	This resource is for <a href="#">Search</a> , <a href="#">Analytics</a> , and <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 event size	64,000,000 bytes	
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	
Profiler data retention	5 days	
Profiler data sent per day	10 GB	

For more information, see [About pricing and quotas in Application Insights](#).

## Azure Policy limits

There's a maximum count for each object type for Azure Policy. An entry of *Scope* means either the subscription or the [management group](#).

WHERE	WHAT	MAXIMUM COUNT
Scope	Policy definitions	500
Scope	Initiative definitions	100
Tenant	Initiative definitions	1,000
Scope	Policy or initiative assignments	100
Policy definition	Parameters	20
Initiative definition	Policies	100
Initiative definition	Parameters	100
Policy or initiative assignments	Exclusions (notScopes)	400
Policy rule	Nested conditionals	512

## 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

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Included messages per unit per day for Standard tier	1,000,000	1,000,000

To request an update to your subscription's default limits, open a support ticket.

## 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>

### 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](#).

<sup>1</sup>To request an increase beyond this limit, contact Azure Support.

## 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

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Reserved IPs per subscription	20	100
Affinity groups 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	DEFAULT LIMIT
Standard sku container groups per region per <a href="#">subscription</a>	100 <sup>1</sup>
Dedicated sku container groups per region per <a href="#">subscription</a>	0 <sup>1</sup>
Number of containers per container group	60
Number of volumes per container group	20
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 creates per hour	300 <sup>1</sup>
Container creates per 5 minutes	100 <sup>1</sup>
Container deletes per hour	300 <sup>1</sup>
Container deletes per 5 minutes	100 <sup>1</sup>

<sup>1</sup>To request a limit increase, create an [Azure Support request](#).

## Container Registry limits

The following table details the features and limits of the Basic, Standard, and Premium [service tiers](#).

RESOURCE	BASIC	STANDARD	PREMIUM
Storage <sup>1</sup>	10 GiB	100 GiB	500 GiB
Maximum image layer size	200 GiB	200 GiB	200 GiB
ReadOps per minute <sup>2, 3</sup>	1,000	3,000	10,000
WriteOps per minute <sup>2, 4</sup>	100	500	2,000

RESOURCE	BASIC	STANDARD	PREMIUM
Download bandwidth MBps <sup>2</sup>	30	60	100
Upload bandwidth MBps <sup>2</sup>	10	20	50
Webhooks	2	10	500
Geo-replication	N/A	N/A	Supported
Content trust	N/A	N/A	Supported
Virtual network access	N/A	N/A	Preview
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>The specified storage limits are the amount of *included* storage for each tier. You're charged an additional daily rate per GiB for image storage above these limits. For rate information, see [Azure Container Registry pricing](#).

<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	DEFAULT LIMIT
Azure Content Delivery Network profiles	25
Content Delivery Network endpoints per profile	25
Custom domains per endpoint	25

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 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
Data factories in an Azure subscription	50	Contact support.
Total number of entities, such as pipelines, data sets, triggers, linked services, and integration runtimes, within a data factory	5,000	Contact support.
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	Contact support.
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, HDInsights, Web, and others.	3000	Contact support.
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.	1000	Contact support.
Concurrent authoring operations per subscription per <a href="#">Azure Integration Runtime region</a> Including test connection, browse folder list and table list, preview data.	200	Contact support.
Concurrent Data Integration Units <sup>1</sup> consumption per subscription per <a href="#">Azure Integration Runtime region</a>	Region group 1 <sup>2</sup> : 6000 Region group 2 <sup>2</sup> : 3000 Region group 3 <sup>2</sup> : 1500	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

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum queued runs per pipeline	100	100
Characters per expression	8,192	8,192
Minimum tumbling window trigger interval	15 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
Data Integration Units <sup>1</sup> per copy activity run	256	<a href="#">Contact support.</a>
Write API calls	1,200/h  This limit is imposed by Azure Resource Manager, not Azure Data Factory.	<a href="#">Contact support.</a>
Read API calls	12,500/h  This limit is imposed by Azure Resource Manager, not Azure Data Factory.	<a href="#">Contact support.</a>
Monitoring queries per minute	1,000	<a href="#">Contact support.</a>
Entity CRUD operations per minute	50	<a href="#">Contact support.</a>
Maximum time of data flow debug session	8 hrs	8 hrs
Concurrent number of data flows per factory	50	<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	<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 US2, 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, Northcentral US, Southcentral US, Southeast Asia, West Central US
Region group 3	Canada Central, East Asia, France Central, Korea Central, UK South

<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.

## 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	<a href="#">Contact support</a> .
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 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	DEFAULT 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 Lake Store limits

Azure Data Lake Storage Gen1 is 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	DEFAULT 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	50

RESOURCE	LIMIT
Maximum number of sent shares per Data Share resource	100
Maximum number of received shares per Data Share resource	100
Maximum number of invitations per sent share	100
Maximum number of share subscriptions per sent share	100
Maximum number of datasets per share	100
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	DEFAULT LIMIT	COMMENTS
Maximum number of services per subscription, per region	2	To request an increase for this limit, contact support.

## Event Grid limits

The following limits apply to Azure Event Grid system topics and custom topics, *not* event domains.

RESOURCE	LIMIT
Custom topics per Azure subscription	100
Event subscriptions per topic	500
Publish rate for a custom topic (ingress)	5,000 events per second per topic
Publish requests	250 per second
Event size	1 MB (charged in as multiple 64-KB events)

The following limits apply to event domains only.

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 per second

RESOURCE	LIMIT
Publish requests	250 per second
Event Domains per Azure Subscription	100

## 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](#).

The following limits are common across basic, standard, and dedicated tiers.

LIMIT	SCOPE	NOTES	VALUE
Number of Event Hubs namespaces per subscription	Subscription	-	100
Number of event hubs per namespace	Namespace	Subsequent requests for creation of a new event hub are rejected.	10
Number of partitions per event hub	Entity	-	32
Maximum size of an event hub name	Entity	-	50 characters
Number of non-epoch receivers per consumer group	Entity	-	5
Maximum throughput units	Namespace	Exceeding the throughput unit limit causes your data to be throttled and generates a <a href="#">server busy exception</a> . To request a larger number of throughput units for a Standard tier, file a <a href="#">support request</a> . Additional throughput units are available in blocks of 20 on a committed purchase basis.	20
Number of authorization rules per namespace	Namespace	Subsequent requests for authorization rule creation are rejected.	12
Number of calls to the GetRuntimeInformation method	Entity	-	50 per second
Number of virtual network (VNet) and IP Config rules	Entity	-	128

### Event Hubs Basic and Standard - quotas and limits

LIMIT	SCOPE	NOTES	BASIC	STANDARD
Maximum size of Event Hubs event	Entity		256 KB	1 MB
Number of consumer groups per event hub	Entity		1	20
Number of AMQP connections per namespace	Namespace	Subsequent requests for additional connections are rejected, and an exception is received by the calling code.	100	5,000
Maximum retention period of event data	Entity		1 day	1-7 days
Apache Kafka enabled namespace	Namespace	Event Hubs namespace streams applications using Kafka protocol	No	Yes
Capture	Entity	When enabled, micro-batches on the same stream	No	Yes

### Event Hubs Dedicated - quotas and limits

The Event Hubs Dedicated offering is billed at a fixed monthly price, with a minimum of 4 hours of usage. The Dedicated tier offers all the features of the Standard plan, but with enterprise scale capacity and limits for customers with demanding workloads.

FEATURE	LIMITS
Bandwidth	20 CUs
Namespaces	50 per CU
Event Hubs	1000 per namespace
Ingress events	Included
Message Size	1 MB
Partitions	2000 per CU
Consumer groups	No limit per CU, 1000 per event hub
Brokered connections	100 K included
Message Retention	90 days, 10 TB included per CU
Capture	Included

## Identity Manager limits

CATEGORY	LIMIT
User-assigned managed identities	<ul style="list-style-type: none"><li>When you create user-assigned managed identities, only alphanumeric characters (0-9, a-z, and A-Z) and the hyphen (-) are supported. For the assignment to a virtual machine or virtual machine scale set to work properly, the name is limited to 24 characters.</li><li>If you use the managed identity virtual machine extension, the supported limit is 32 user-assigned managed identities. Without the managed identity virtual machine extension, the supported limit is 512 user-assigned identities.</li></ul>

## 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	100
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

RESOURCE	LIMIT
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)
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 100 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)
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

The following table lists the limits that apply to Azure IoT Hub Device Provisioning Service resources.

RESOURCE	LIMIT
Maximum device provisioning services per Azure subscription	10
Maximum number of enrollments	1,000,000
Maximum number of registrations	1,000,000
Maximum number of enrollment groups	100
Maximum number of CAs	25
Maximum number of linked IoT hubs	50
Maximum size of message	96 KB

**NOTE**

To increase the number of enrollments and registrations on your provisioning service, contact [Microsoft Support](#).

**NOTE**

Increasing the maximum number of CAs is not supported.

The Device Provisioning Service throttles requests when the following quotas are exceeded.

THROTTLE	PER-UNIT VALUE
Operations	200/min/service
Device registrations	200/min/service
Device polling operation	5/10 sec/device

## Key Vault limits

**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

KEY TYPE	HSM KEY CREATE KEY	HSM KEY ALL OTHER TRANSACTIONS	SOFTWARE KEY CREATE KEY	SOFTWARE KEY ALL OTHER TRANSACTIONS
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.

## 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.

RESOURCE	DEFAULT LIMIT
Azure Media Services accounts in a single subscription	25 (fixed)
Media reserved units per Media Services account	25 (S1) 10 (S2, S3) <sup>1</sup>
Jobs per Media Services account	50,000 <sup>2</sup>
Chained tasks per job	30 (fixed)

RESOURCE	DEFAULT LIMIT
Assets per Media Services account	1,000,000
Assets per task	50
Assets per job	100
Unique locators associated with an asset at one time	5 <sup>4</sup>
Live channels per Media Services account	5
Programs in stopped state per channel	50
Programs in running state per channel	3
Streaming endpoints that are stopped or running per Media Services account	2
Streaming units per streaming endpoint	10
Storage accounts	1,000 <sup>5</sup> (fixed)
Policies	1,000,000 <sup>6</sup>
File size	In some scenarios, there's a limit on the maximum file size supported for processing in Media Services. <sup>7</sup>

<sup>1</sup>If you change the type, for example, from S2 to S1, the maximum reserved unit limits are reset.

<sup>2</sup>This number includes queued, finished, active, and canceled jobs. It doesn't include deleted jobs. You can delete old jobs by using **IJob.Delete** or the **DELETE** HTTP request.

As of April 1, 2017, any job record in your account older than 90 days is automatically deleted, along with its associated task records. Automatic deletion occurs even if the total number of records is below the maximum quota. To archive the job and task information, use the code described in [Manage assets with the Media Services .NET SDK](#).

<sup>3</sup>When you make a request to list job entities, a maximum of 1,000 jobs is returned per request. To keep track of all submitted jobs, use the top or skip queries as described in [OData system query options](#).

<sup>4</sup>Locators aren't designed for managing per-user access control. To give different access rights to individual users, use digital rights management (DRM) solutions. For more information, see [Protect your content with Azure Media Services](#).

<sup>5</sup>The storage accounts must be from the same Azure subscription.

<sup>6</sup>There's a limit of 1,000,000 policies for different Media Services policies. An example is for the Locator policy or ContentKeyAuthorizationPolicy.

#### NOTE

If you always use the same days and access permissions, use the same policy ID. For information and an example, see [Manage assets with the Media Services .NET SDK](#).

<sup>7</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.

The following table shows the limits on the media reserved units S1, S2, and S3. If your source file is larger than the limits defined in the table, your encoding job fails. If you encode 4K resolution sources of long duration, you're required to use S3 media reserved units to achieve the performance needed. If you have 4K content that's larger than the 260-GB limit on the S3 media reserved units, open a support ticket.

MEDIA RESERVED UNIT TYPE	MAXIMUM INPUT SIZE (GB)
S1	26
S2	60
S3	260

## 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
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 <b>X</b> 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 <b>X</b> consecutive MFA denials	Not set	99
Reset account lockout counter after <b>X</b> minutes	Not set	9,999
Unlock account after <b>X</b> 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	DEFAULT/MAXIMUM 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

RESOURCE	DEFAULT/MAXIMUM LIMIT
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
User-defined route tables	200
User-defined routes per route table	400
Point-to-site root certificates per Azure VPN Gateway	20
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</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 Prefixes	limited by number of Standard Public IPs in a subscription	Contact support.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
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.

#### 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	DEFAULT/MAXIMUM LIMIT
Load balancers	1,000
Rules per resource	1,500
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations	600
Backend pool size	1,000 IP configurations, single virtual network
High-availability ports	1 per internal frontend
Outbound rules per Load Balancer	20

#### Basic Load Balancer

RESOURCE	DEFAULT/MAXIMUM LIMIT
Load balancers	1,000
Rules per resource	250
Rules per NIC (across all IPs on a NIC)	300
Frontend IP configurations	200
Backend pool size	300 IP configurations, single availability set
Availability sets per Load Balancer	150

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

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
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.
Network Security Groups (NSGs)	200	200
NSG rules per NSG	1,000	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 VIP per deployment	5	Contact support
Private VIP (internal load balancing) per deployment	1	1
Endpoint access control lists (ACLs)	50	50

## ExpressRoute limits

RESOURCE	DEFAULT/MAXIMUM LIMIT
ExpressRoute circuits per subscription	10
ExpressRoute circuits per region per subscription, with Azure Resource Manager	10
Maximum number of routes advertised to Azure private peering with ExpressRoute Standard	4,000
Maximum number of routes advertised to Azure private peering with ExpressRoute Premium add-on	10,000
Maximum number of routes advertised from Azure private peering from the VNet address space for an ExpressRoute connection	200
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	DEFAULT/MAXIMUM 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	4
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 WAN limits

RESOURCE	LIMIT
Virtual WAN hubs per region	1
Virtual WAN hubs per virtual wan	Azure regions
VPN (branch) connections per hub	1,000

RESOURCE	LIMIT
VNet connections per hub	500
Point-to-Site users per hub	10,000
Aggregate throughput per Virtual WAN VPN gateway	20 Gbps
Throughput per Virtual WAN VPN connection (2 tunnels)	2 Gbps with 1 Gbps/IPsec tunnel
Aggregate throughput per Virtual WAN ExpressRoute gateway	20 Gbps

## Application Gateway limits

The following table applies to v1, v2, Standard, and WAF SKUs unless otherwise stated.

RESOURCE	DEFAULT/MAXIMUM LIMIT	NOTE
Azure Application Gateway	1,000 per subscription	
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	100 <sup>1</sup>	
HTTP load-balancing rules	100 <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	

RESOURCE	DEFAULT/MAXIMUM LIMIT	NOTE
Maximum path-based rules per URL map	100	
Redirect configurations	100 <sup>1</sup>	
Concurrent WebSocket connections	Medium gateways 20k Large gateways 50k	
Maximum URL length	32KB	
Maximum header size for HTTP/2	4KB	
Maximum file upload size, Standard	2 GB	
Maximum file upload size WAF	V1 Medium WAF gateways, 100 MB V1 Large WAF gateways, 500 MB V2 WAF, 750 MB	
WAF body size limit, without files	128 KB	
Maximum WAF custom rules	100	
Maximum WAF exclusions	100	

<sup>1</sup> In case of WAF-enabled SKUs, we recommend that you limit the number of resources to 40 for optimal performance.

## Network Watcher limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT	NOTE
Azure Network Watcher	1 per region	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	10,000	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 service per subscription	800
Number of IP Configurations on a private link service	8 (This number is for the NAT IP addresses used per PLS)

RESOURCE	LIMIT
Number of private endpoints on the same private link service	1000

### Traffic Manager limits

RESOURCE	DEFAULT/MAXIMUM LIMIT
Profiles per subscription	200
Endpoints per profile	200

### Azure Bastion limits

RESOURCE	DEFAULT LIMIT
Concurrent RDP connections	25*
Concurrent SSH connections	More than 50**

\*May vary due to other on-going RDP sessions or other on-going SSH sessions.

\*\*May vary if there are existing RDP connections or usage from other on-going SSH sessions.

### Azure DNS limits

#### Public DNS zones

RESOURCE	DEFAULT 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
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

RESOURCE	DEFAULT LIMIT
Number of DNS queries a virtual machine can send to Azure DNS resolver, per second	500 <sup>2</sup>
Maximum number of DNS queries queued (pending response) per virtual machine	200 <sup>2</sup>

<sup>1</sup>If you need to increase these limits, contact Azure Support.

<sup>2</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	DEFAULT LIMIT
Data throughput	30 Gbps <sup>1</sup>
Rules	10,000. All rule types combined.
Maximum DNAT rules	299
Minimum AzureFirewallSubnet size	/26
Port range in network and application rules	0-64,000. Work is in progress to relax this limitation.
Public IP addresses	100 maximum (Currently, SNAT ports are added only for the first five public IP addresses.)
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>

<sup>1</sup>If you need to increase these limits, contact Azure Support.

## Azure Front Door Service limits

RESOURCE	DEFAULT/MAXIMUM LIMIT
Azure Front Door Service resources per subscription	100
Front-end hosts, which includes custom domains per resource	100

RESOURCE	DEFAULT/MAXIMUM LIMIT
Routing rules per resource	100
Back-end pools per resource	50
Back ends per back-end pool	100
Path patterns to match for a routing rule	25
Custom web application firewall rules per policy	10
Web application firewall policy per subscription	100
Web application firewall match conditions per custom rule	10
Web application firewall IP address ranges per match condition	600
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 HTTP request body size inspected	128 KB
Web application firewall custom response body length	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.
- 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.
- Front Door to the back-end TCP session timeout is 30 minutes.

## Upload and download data limit

	WITH CHUNKED TRANSFER ENCODING (CTE)	WITHOUT HTTP CHUNKING
<b>Download</b>	There's no limit on the download size.	There's no limit on the download size.

	WITH CHUNKED TRANSFER ENCODING (CTE)	WITHOUT HTTP CHUNKING
<b>Upload</b>	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.

## 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](#).

## Role-based access control limits

RESOURCE	LIMIT
Role assignments for Azure resources per Azure subscription	2,000
Role assignments for Azure resources per management group	500
Custom roles for Azure resources per tenant	5,000
Custom roles for Azure resources per tenant (specialized clouds, such as Azure Government, Azure Germany, and Azure China 21Vianet)	2,000

## 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	NOTES	VALUE
Maximum number of Basic or Standard namespaces per Azure subscription	Namespace	Subsequent requests for additional Basic or Standard namespaces are rejected by the Azure portal.	100

Quota name	Scope	Notes	Value
Maximum number of Premium namespaces per Azure subscription	Namespace	Subsequent requests for additional Premium namespaces are rejected by the portal.	100
Queue or topic size	Entity	Defined upon creation of the queue or topic.  Subsequent incoming messages are rejected, and an exception is received by the calling code.	1, 2, 3, 4 GB or 5 GB.  In the Premium SKU, and the Standard SKU with <a href="#">partitioning</a> enabled, the maximum queue or topic size is 80 GB.
Number of concurrent connections on a namespace	Namespace	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.	NetMessaging: 1,000.  AMQP: 5,000.
Number of concurrent receive requests on a queue, topic, or subscription entity	Entity	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.	5,000
Number of topics or queues per namespace	Namespace	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.	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). Maximum limit is 4,000.
Number of <a href="#">partitioned topics or queues</a> per namespace	Namespace	Subsequent requests for creation of a new partitioned 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, the exception <b>QuotaExceededException</b> is received by the calling code.	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.
Maximum size of any messaging entity path: queue or topic	Entity	-	260 characters.

Quota Name	Scope	Notes	Value
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
Message size for a queue, topic, or subscription entity	Entity	<p>Incoming messages that exceed these quotas are rejected, and an exception is received by the calling code.</p>	<p>Maximum message size: 256 KB for <a href="#">Standard tier</a>, 1 MB for <a href="#">Premium tier</a>.</p> <p>Due to system overhead, this limit is less than these values.</p> <p>Maximum header size: 64 KB.</p> <p>Maximum number of header properties in property bag: <a href="#">byte/int.MaxValue</a>.</p> <p>Maximum size of property in property bag: No explicit limit. Limited by maximum header size.</p>
Message property size for a queue, topic, or subscription entity	Entity	The exception <a href="#">SerializationException</a> is generated.	Maximum message property size for each property is 32,000. Cumulative size of all properties can't exceed 64,000. This limit applies to the entire header of the <a href="#">BrokeredMessage</a> , which has both user properties and system properties, such as <a href="#">SequenceNumber</a> , <a href="#">Label</a> , and <a href="#">MessageId</a> .
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.
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

Quota Name	Scope	Notes	Value
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
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 <a href="#">SharedAccessAuthorizationRule</a> 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 "Cannot send more than 100 messages in a single transaction" is received by the calling code.	100  For both <b>Send()</b> and <b>SendAsync()</b> operations.
Number of virtual network and IP filter rules	Namespace		128

## Site Recovery limits

The following limits apply to Azure Site Recovery.

Limit Identifier	Default Limit
Number of vaults per subscription	500
Number of servers per Azure vault	250
Number of protection groups per Azure vault	No limit
Number of recovery plans per Azure vault	No limit
Number of servers per protection group	No limit
Number of servers per recovery plan	50

## 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 managed instances](#).

## SQL Data Warehouse limits

For SQL Data Warehouse limits, see [SQL Data Warehouse resource limits](#).

## Storage limits

The following table describes default limits for Azure general-purpose v1, v2, and Blob storage accounts. The *ingress* limit refers to all data from requests that are sent to a storage account. The *egress* limit refers to all data from responses that are received from a storage account.

RESOURCE	DEFAULT LIMIT
Number of storage accounts per region per subscription, including both standard and premium accounts	250
Maximum storage account capacity	2 PiB for US and Europe, and 500 TiB for all other regions (including the UK) <sup>1</sup>
Maximum number of blob containers, blobs, file shares, tables, queues, entities, or messages per storage account	No limit
Maximum request rate <sup>1</sup> per storage account	20,000 requests per second
Maximum ingress <sup>1</sup> per storage account (US, Europe regions)	25 Gbps
Maximum ingress <sup>1</sup> per storage account (regions other than US and Europe)	5 Gbps if RA-GRS/GRS is enabled, 10 Gbps for LRS/ZRS <sup>2</sup>
Maximum egress for general-purpose v2 and Blob storage accounts (all regions)	50 Gbps
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 virtual network rules per storage account	200
Maximum number of IP address rules per storage account	200

<sup>1</sup>Azure Storage standard accounts support higher capacity limits and higher limits for ingress by request. To request an increase in account limits for ingress, contact [Azure Support](#). For more information, see [Announcing larger, higher scale storage accounts](#).

<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 those of the primary location. [Azure Storage replication](#) options include:

- [Locally redundant storage \(LRS\)](#)
- [Zone-redundant storage \(ZRS\)](#)

- [Geo-redundant storage \(GRS\)](#)
- [Read-access geo-redundant storage \(RA-GRS\)](#)
- [Geo-zone-redundant storage \(GZRS\)](#)
- [Read-access geo-zone-redundant storage \(RA-GZRS\)](#)

**NOTE**

Microsoft recommends that you use a general-purpose v2 storage account for most scenarios. You can easily upgrade a general-purpose v1 or an Azure Blob storage account to a general-purpose v2 account with no downtime and without the need to copy data. For more information, see [Upgrade to a general-purpose v2 storage account](#).

If the needs of your application exceed the scalability targets of a single storage account, you can build your application to use multiple storage accounts. You can then partition your data objects across those storage accounts. For information on volume pricing, see [Azure Storage pricing](#).

All storage accounts run on a flat network topology and support the scalability and performance targets outlined in this article, regardless of when they were created. For more information on the Azure Storage flat network architecture and on scalability, see [Microsoft Azure Storage: A Highly Available Cloud Storage Service with Strong Consistency](#).

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	DEFAULT LIMIT
Storage account management operations (read)	800 per 5 minutes
Storage account management operations (write)	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	100 MiB
Maximum size of a block blob	50,000 X 100 MiB (approximately 4.75 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
Maximum number of stored access policies per blob container	5

RESOURCE	TARGET
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
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.

## Azure Files limits

For more information on Azure Files limits, see [Azure Files scalability and performance targets](#).

RESOURCE	STANDARD FILE SHARES	PREMIUM FILE SHARES
Minimum size of a file share	No minimum; pay as you go	100 GiB; provisioned
Maximum size of a file share	100 TiB*, 5 TiB	100 TiB
Maximum size of a file in a file share	1 TiB	1 TiB
Maximum number of files in a file share	No limit	No limit
Maximum IOPS per share	10,000 IOPS*, 1,000 IOPS	100,000 IOPS
Maximum number of stored access policies per file share	5	5
Target throughput for a single file share	up to 300 MiB/sec*, Up to 60 MiB/sec ,	See premium file share ingress and egress values
Maximum egress for a single file share	See standard file share target throughput	Up to 6,204 MiB/s
Maximum ingress for a single file share	See standard file share target throughput	Up to 4,136 MiB/s
Maximum open handles per file	2,000 open handles	2,000 open handles
Maximum number of share snapshots	200 share snapshots	200 share snapshots
Maximum object (directories and files) name length	2,048 characters	2,048 characters
Maximum pathname component (in the path \A\B\C\D, each letter is a component)	255 characters	255 characters

\* Available in most regions, see [Regional availability](#) for the details on available regions.

## Azure File Sync limits

RESOURCE	TARGET	HARD LIMIT
Storage Sync Services per region	20 Storage Sync Services	Yes
Sync groups per Storage Sync Service	100 sync groups	Yes
Registered servers per Storage Sync Service	99 servers	Yes
Cloud endpoints per sync group	1 cloud endpoint	Yes
Server endpoints per sync group	50 server endpoints	No
Server endpoints per server	30 server endpoints	Yes
File system objects (directories and files) per sync group	100 million objects	No
Maximum number of file system objects (directories and files) in a directory	5 million objects	Yes
Maximum object (directories and files) security descriptor size	64 KiB	Yes
File size	100 GiB	No
Minimum file size for a file to be tiered	V9: Based on file system cluster size (double file system cluster size). For example, if the file system cluster size is 4kb, the minimum file size will be 8kb. V8 and older: 64 KiB	Yes

### NOTE

An Azure File Sync endpoint can scale up to the size of an Azure file share. If the Azure file share size limit is reached, sync will not be able to operate.

## 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

RESOURCE	TARGET
Maximum size of a single table	500 TiB
Maximum size of a table entity	1 MiB
Maximum number of properties in a table entity	255, which includes three system properties: PartitionKey, RowKey, and Timestamp
Maximum total size of property values in an entity	1 MiB
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> .
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. 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. There is no limit for the number of Managed Disks, snapshots and images per resource group.

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
Standard managed disks	50,000	50,000
Standard SSD managed disks	50,000	50,000
Premium managed disks	50,000	50,000
Standard_LRS snapshots	50,000	50,000
Standard_ZRS snapshots	50,000	50,000
Managed image	50,000	50,000

- **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](#).

## Managed virtual machine disks

### Standard HDD managed disks

STAND ARD 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 MiB/sec	Up to 300 MiB/sec	Up to 500 MiB/sec	Up to 500 MiB/sec							

### Standard SSD managed disks

STA NDAR SSD SIZE S	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 120	Up to 120	Up to 120	Up to 120	Up to 240	Up to 500	Up to 2,000	Up to 4,000	Up to 6,000					
Throughput per disk	Up to 25 MiB/sec	Up to 50 MiB/sec	Up to 60 MiB/sec	Up to 400 MiB/sec	Up to 600 MiB/sec	Up to 750 MiB/sec								

\*Denotes a disk size that is currently in preview, for regional availability information see [New disk sizes: Managed and unmanaged](#).

## Premium SSD managed disks: Per-disk limits

PRE MIU M SSD SIZE S	P1*	P2*	P3*	P4	P6	P10	P15	P20	P30	P40	P50	P60	P70	P80
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	120	120	120	120	240	500	1,100	2,300	5,000	7,500	7,500	16,000	18,000	20,000
Throughput per disk	25 MiB/sec	25 MiB/sec	25 MiB/sec	25 MiB/sec	50 MiB/sec	100 MiB/sec	125 MiB/sec	150 MiB/sec	200 MiB/sec	250 MiB/sec	250 MiB/sec	500 MiB/sec	750 MiB/sec	900 MiB/sec
Max burst IOPS per disk **	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500	3,500
Max burst throughput per disk **	170 MiB/sec	170 MiB/sec	170 MiB/sec	170 MiB/sec	170 MiB/sec	170 MiB/sec								
Max burst duration**	30 min	30 min	30 min	30 min	30 min	30 min								
Eligible for reservation	No	Yes, up to one year												

\*Denotes a disk size that is currently in preview, for regional availability information see [New disk sizes: Managed and unmanaged](#).

\*\*Denotes a feature that is currently in preview, see [Disk bursting](#) for more information.

## Premium SSD managed disks: Per-VM limits

RESOURCE	DEFAULT 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	DEFAULT 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)
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	DEFAULT 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	
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	

LIMIT IDENTIFIER	LIMIT	COMMENTS
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>
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	300	Reference data blobs can't be larger than 300 MB each.

## Virtual Machines limits

### Virtual Machines limits

RESOURCE	DEFAULT LIMIT	MAXIMUM LIMIT
<a href="#">Virtual machines</a> per cloud service <sup>1</sup>	50	50
Input endpoints per cloud service <sup>2</sup>	150	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. For more information, see [How to set up endpoints to a virtual machine](#).

### Virtual Machines limits - Azure Resource Manager

The following limits apply when you use Azure Resource Manager and Azure resource groups.

RESOURCE	DEFAULT 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.
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.

RESOURCE	DEFAULT LIMIT
Availability sets per subscription	2,000 per region.
Virtual machines per availability set	200
Certificates per subscription	Unlimited <sup>2</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.

<sup>2</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	DEFAULT LIMIT	MAXIMUM LIMIT
Maximum number of VMs in a scale set	1,000	1,000
Maximum number of VMs based on a custom VM image in a scale set	600	600
Maximum number of scale sets in a region	2,000	2,000

## 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](#)

# Best Practices for Azure App Service

1/8/2020 • 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}/resourceGroups/{resourceGroupName}/providers/Microsoft.Web/sites/{siteName}/diagnosticLogs](https://ms.portal.azure.com/?websitesextension_ext=asd.featurePath%3Ddetectors%2FParentAvailabilityAndPerformance#@microsoft.onmicrosoft.com/resource/subscriptions/{subscriptionId}/resourceGroups/{resourceGroupName}/providers/Microsoft.Web/sites/{siteName}/diagnosticLogs)

# Troubleshoot an app in Azure App Service using Visual Studio

12/2/2019 • 29 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**.

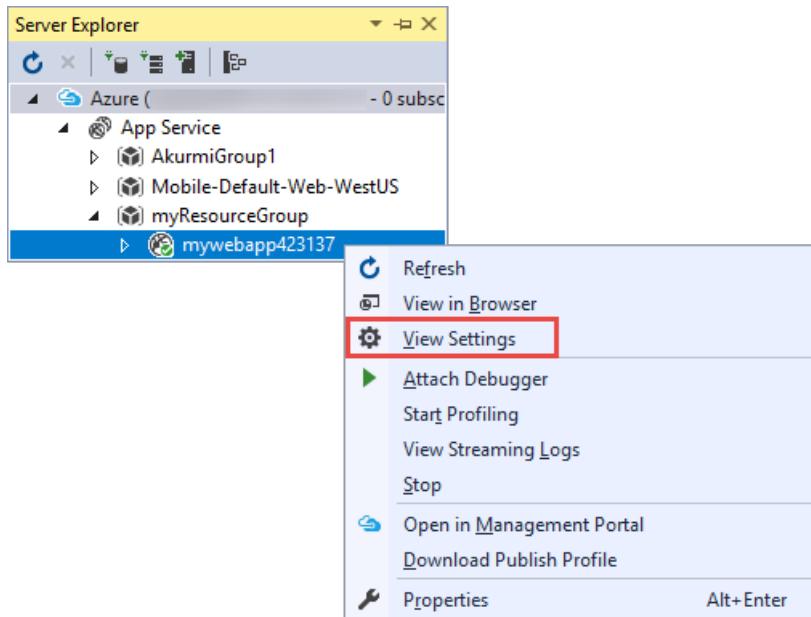
An alternative is to install a management certificate that enables access to your account. If you choose to install a certificate, right-click the **Azure** node in **Server Explorer**, and then select **Manage and Filter Subscriptions** in the context menu. In the **Manage Microsoft Azure Subscriptions** 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 [Manage Accounts, Subscriptions, and Administrative Roles](#).

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.

The screenshot shows the 'Configuration' blade for a web app named 'mywebapp423137'. At the top, there are 'Save' and 'Refresh' buttons. Below them is a 'Actions' section with three options: 'Open in Management Portal', 'Stop Web App', and 'Restart Web App'. The main area is divided into three sections: 'Web App Settings', 'Connection Strings', and 'Application Settings'.  
**Web App Settings:** This section contains six dropdowns:

- .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:** This section contains a table with columns 'Name', 'Connection String', and 'Database Type'. It has an 'Add' button at the bottom.

Name	Connection String	Database Type

**Application Settings:** This section contains a table with columns 'Name' and 'Value'. It has an 'Add' button at the bottom.

Name	Value
WEBSITE_NODE_INDEX	6.9.1

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:**

**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:

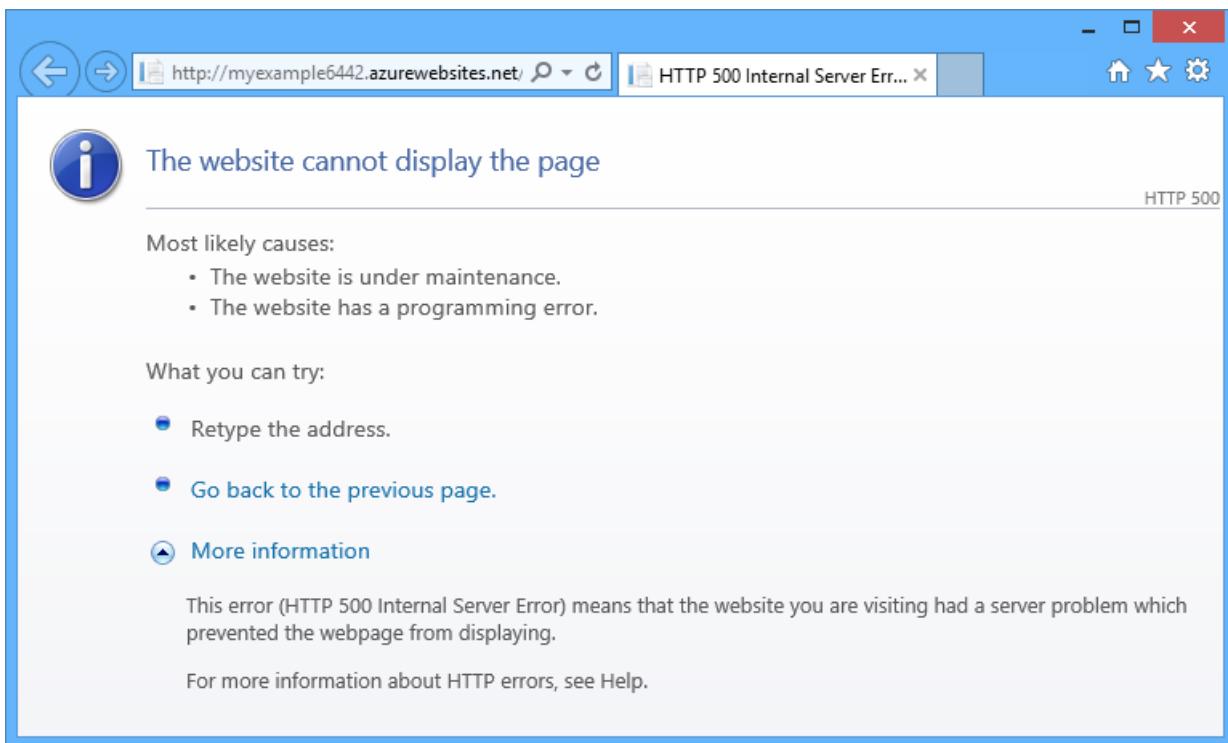
Application name

Error.

An error occurred while processing your request.

© 2014 - My ASP.NET Application

The website cannot display the page

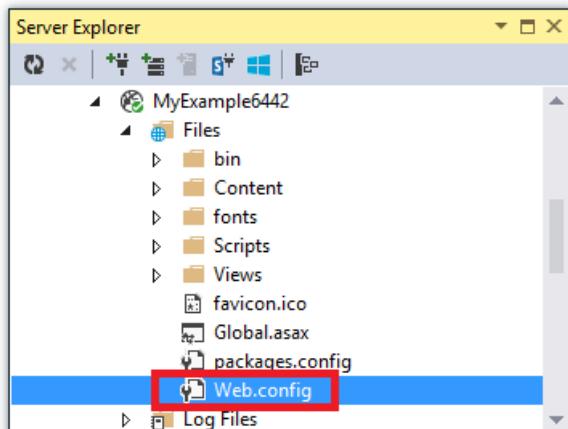


Frequently the easiest way to find the cause of the error is to enable detailed error messages, which the first of the preceding screenshots explains how to do. That requires a change in the deployed Web.config file. You could edit the *Web.config* file in the project and redeploy the project, or create 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.

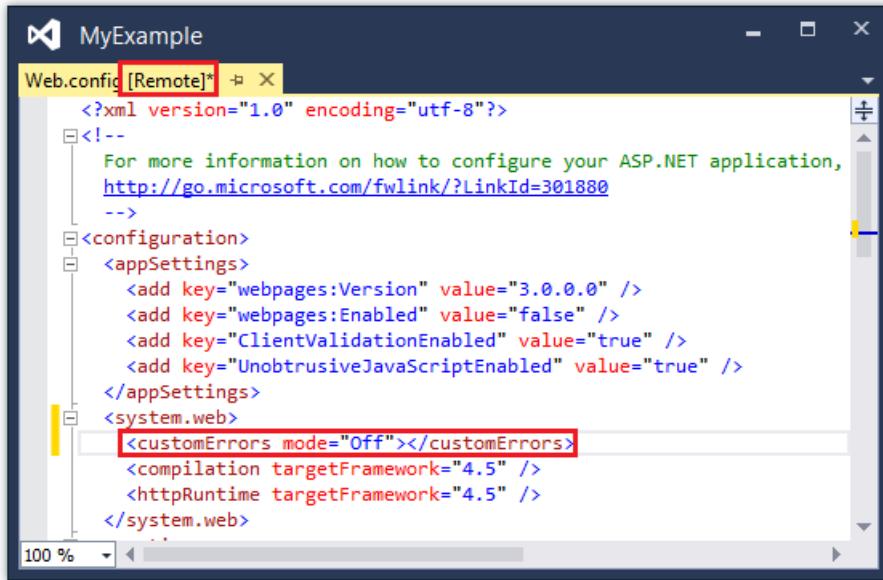
2. Expand the **Files** node, and double-click the *Web.config* file.



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:

A screenshot of a browser window showing a server error. The title bar says 'http://myexample6442.azurewebsites...'. The main content area has a red background and displays:

**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.

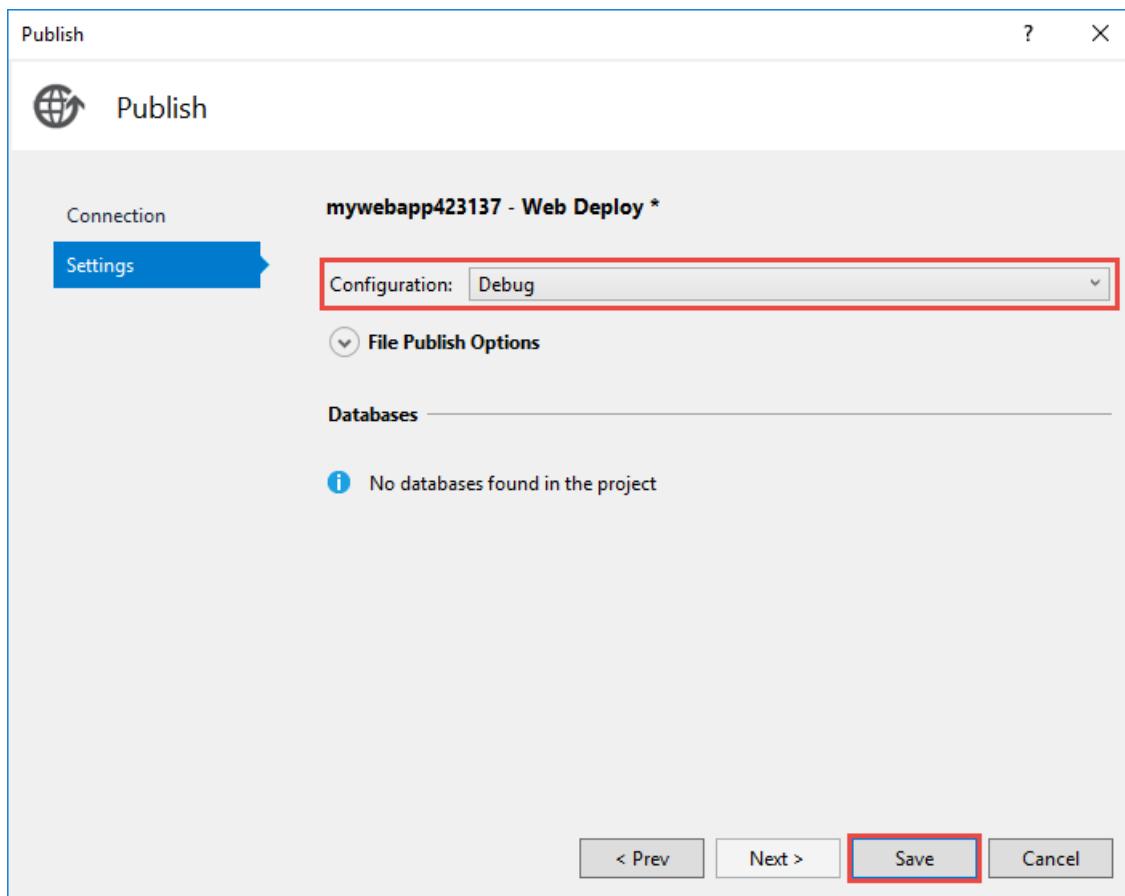
Remote debugging does not work in Express editions of Visual Studio.

This section shows how to debug remotely using the project you create in [Create an ASP.NET app in Azure App Service](#).

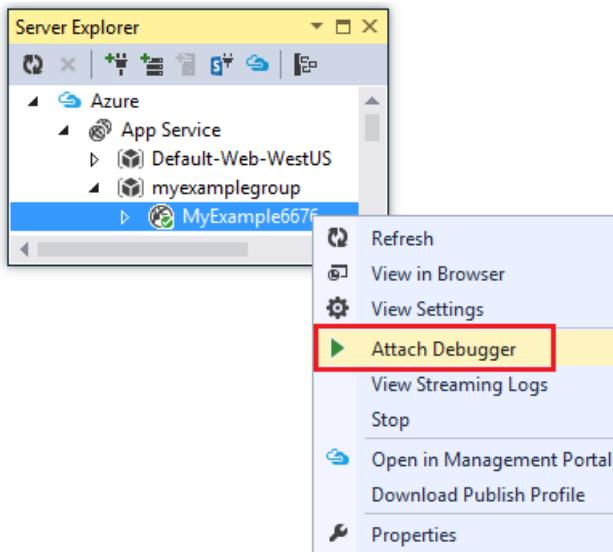
1. Open the web project that you created in [Create an ASP.NET app in Azure App Service](#).
2. Open *Controllers\HomeController.cs*.
3. Delete the `About()` method and insert the following code in its place.

```
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("T");
    ViewBag.Message = "The current time is " + currentTime;

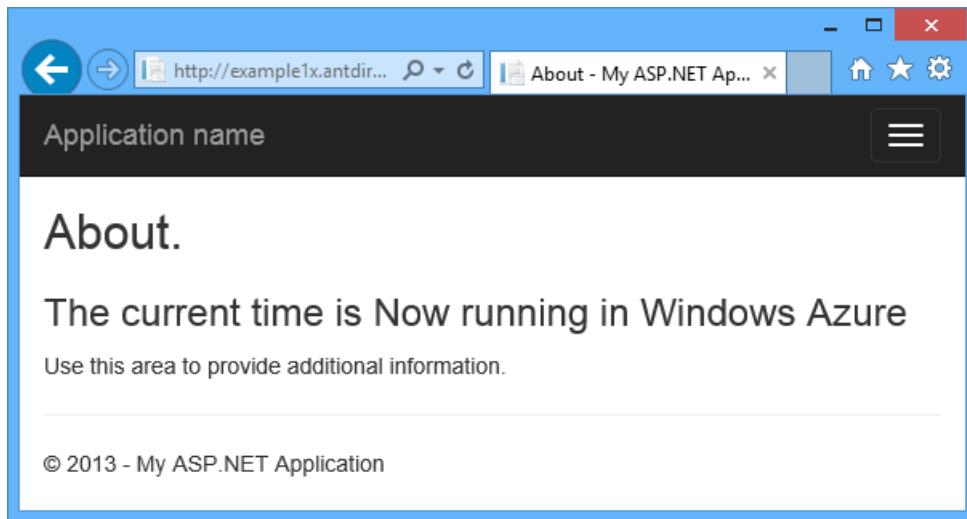
    return View();
}
```

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](#).

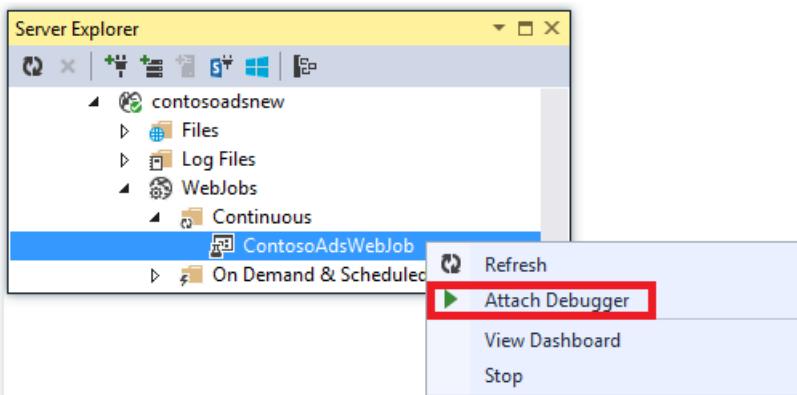
The features shown in this section are available only in Visual Studio 2013 with Update 4 or later.

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 `GenerateThumbnail` method.

```
public class Functions
{
    public static void GenerateThumbnail(
        [QueueTrigger("thumbnailrequest")] BlobInformation blobInfo,
        [Blob("images/{BlobName}", FileAccess.Read)] Stream input,
        [Blob("images/{BlobNameWithoutExtension}_thumbnail.jpg")] CloudBlockBlob outputBlob)
    {
        using (Stream output = outputBlob.OpenWrite())
    }
}
```

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**.

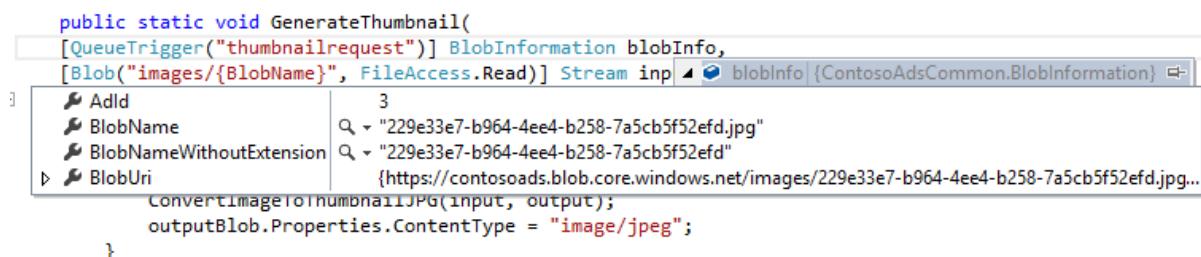


The browser automatically opens to your home page running in Azure. You might have to wait 20 seconds or so while Azure sets up the server for debugging. This delay only happens the first time you run in debug mode on an app in a 48-hour period. When you start debugging again in the same period, there 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.

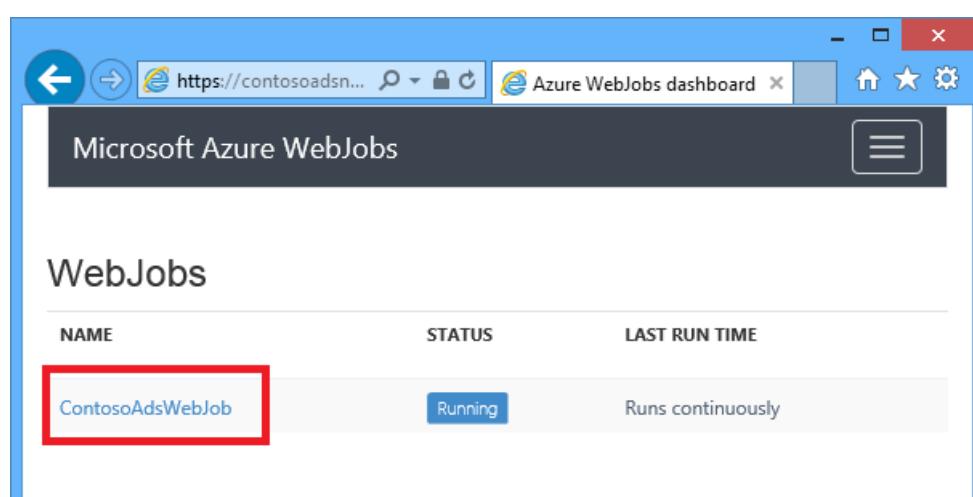
10. When the debugger breaks at your breakpoint, you can examine and change variable values while the program is running in the cloud. In the following illustration, the debugger shows the contents of the blobInfo object that was passed to the `GenerateThumbnail` method.



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 Microsoft Azure WebJobs dashboard. The URL in the address bar is `https://contosoadsns... Azure WebJobs dashboard`. The main title is "Microsoft Azure WebJobs". Below it, the navigation path is "WebJobs / ContosoAdsWebJob / Functions.GenerateThumbnail". The main content area is titled "Invocation Details" and shows the function name "Functions.GenerateThumbnail ({\"BlobUri\":\"https: ...")". A blue button labeled "Replay Function" is visible. A green box indicates a "Success 9 minutes ago (54 seconds running time)". Below this, a note says "⚡ New queue message detected on 'thumbnailrequest'." A table lists parameters and their values:

PARAMETER	VALUE	NOTES
<b>blobInfo</b>	{"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}	
<b>input</b>	images/1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5.jpg	Read 116,652 bytes (100.15% of total). (about 596 milliseconds spent on I/O)
<b>outputBlob</b>	images/1c4a0522-3b73-42cb-ae5e-f3d2cd0d98a5_thumbnail.jpg	

A blue button at the bottom left says "Toggle Output".

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();
}

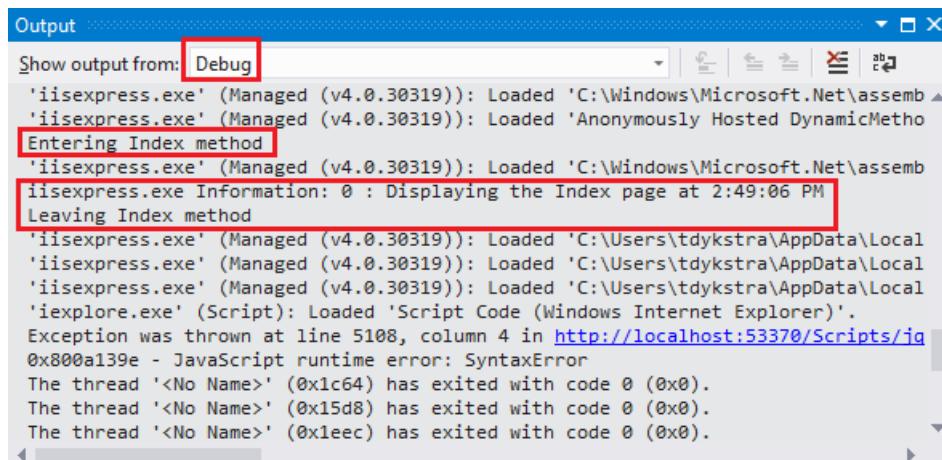
public ActionResult Contact()
{
    Trace.WriteLine("Entering Contact method");
    ViewBag.Message = "Your contact page.";
    Trace.LogError("Fatal error on the Contact page at " + DateTime.Now.ToString());
    Trace.WriteLine("Leaving Contact method");
    return View();
}
```

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 Visual Studio Output window. The title bar says "Output". The dropdown menu "Show output from: Debug" is highlighted with a red box. The main pane displays the following text:

```
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Windows\Microsoft.Net\assembly\iisexpress.exe' (Managed (v4.0.30319)): Loaded 'Anonymously Hosted DynamicMethod' [Entering Index method]
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Windows\Microsoft.Net\assembly\iisexpress.exe' Information: 0 : Displaying the Index page at 2:49:06 PM [Leaving Index method]
'iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local\iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local\iisexpress.exe' (Managed (v4.0.30319)): Loaded 'C:\Users\tdykstra\AppData\Local\iexplore.exe' (Script): Loaded 'Script Code (Windows Internet Explorer)'.
Exception was thrown at line 5108, column 4 in http://localhost:53370/Scripts/ja
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:

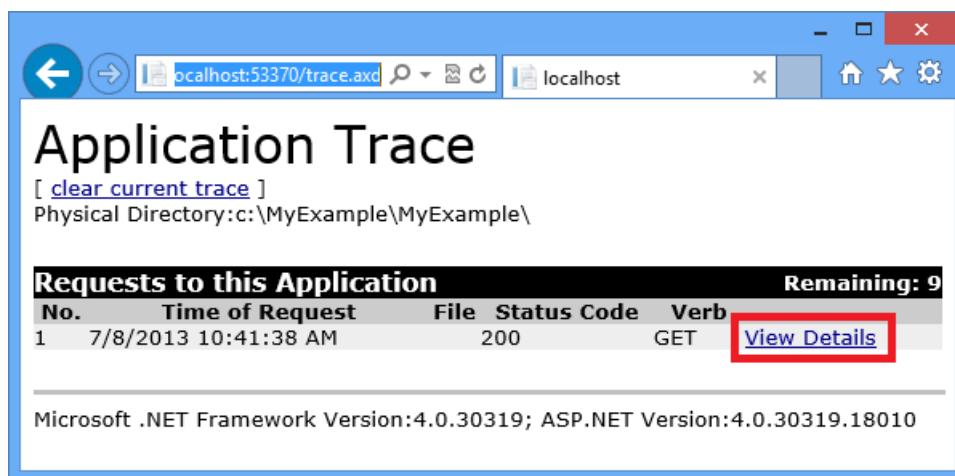
```
<system.diagnostics>
<trace>
  <listeners>
    <add name="WebPageTraceListener"
      type="System.Web.WebPageTraceListener,
      System.Web,
      Version=4.0.0.0,
      Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" />
  </listeners>
</trace>
</system.diagnostics>
```

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.

**Request Details**

Session Id:	Request Type:	GET
Time of Request:	Status Code:	200
Request Encoding:	Response Encoding:	Unicode (UTF-8)

**Trace Information**

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

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.

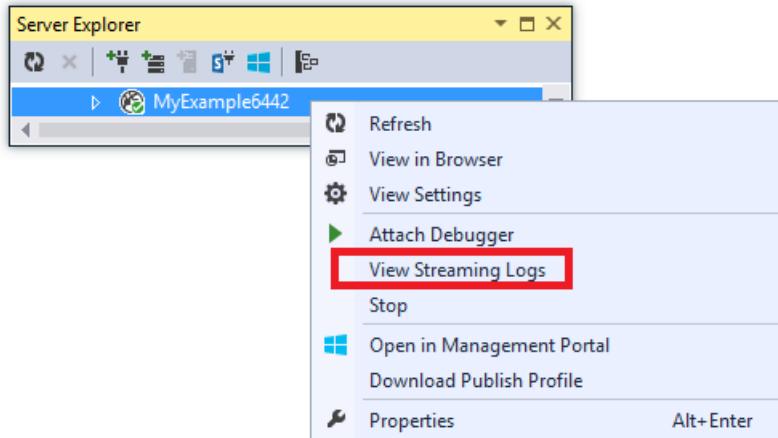
### View the tracing output in Azure

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.

The screenshot shows the Visual Studio Output window with the title 'Output' at the top. Below it, a dropdown menu says 'Show output from: Windows Azure Logs - example1z'. The main area displays log messages:

```

Connecting to Application logs ...
2013-07-08T18:16:39 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T18:17:39 No new trace in the past 1 min(s).
Application: 2013-07-08T18:18:39 No new trace in the past 2 min(s).

```

At the bottom of the window, there are two tabs: 'Web Publish Activity' and 'Output', with 'Output' being the active tab.

- 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.

The screenshot shows the Visual Studio Output window with the title 'Output' at the top. Below it, a dropdown menu says 'Show output from: Windows Azure Logs - example1z'. The main area displays log messages, with the last entry highlighted by a red box:

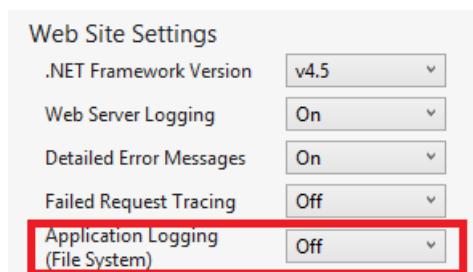
```

Connecting to Application logs ...
2013-07-08T19:00:46 Welcome, you are now connected to log-streaming service.
Application: 2013-07-08T19:01:08 PID[3268] Error Fatal error on the
Contact page at 7:01:08 PM

```

At the bottom of the window, there are two tabs: 'Web Publish Activity' and 'Output', with 'Output' being the active tab.

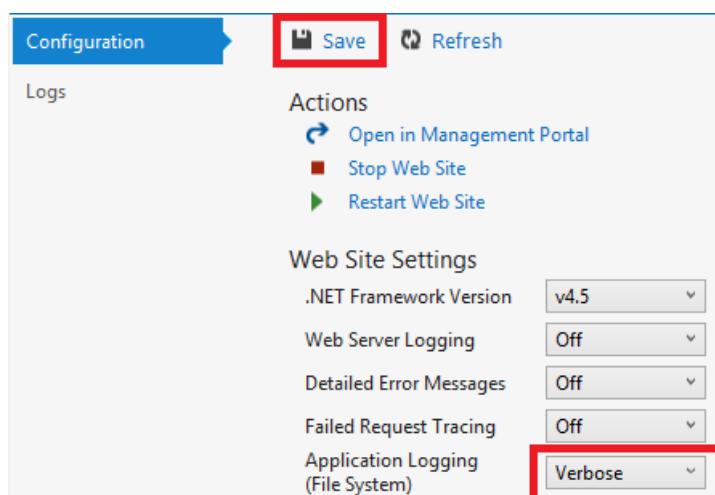
Visual Studio is only showing error-level traces because that is the default setting when you enable the log monitoring service. When you create a new App Service app, all logging is disabled by default, as you saw when you opened the settings page earlier:



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.

- In **Server Explorer**, right-click the app, and then click **View Settings** as you did earlier.

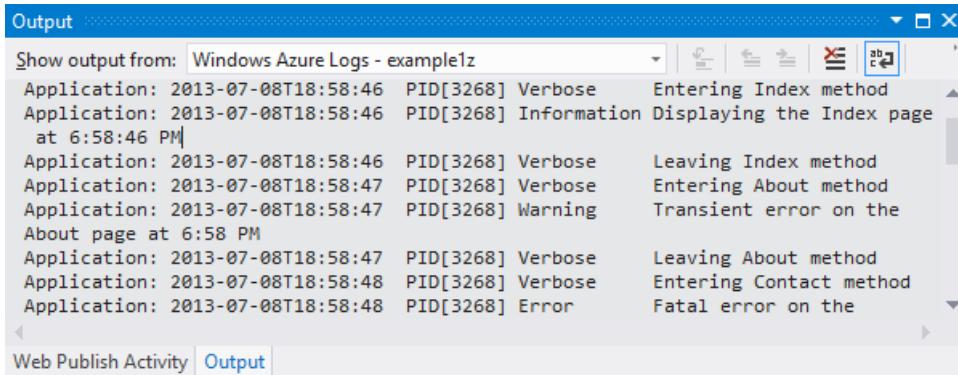
- Change **Application Logging (File System)** to **Verbose**, and then click **Save**.



- 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.



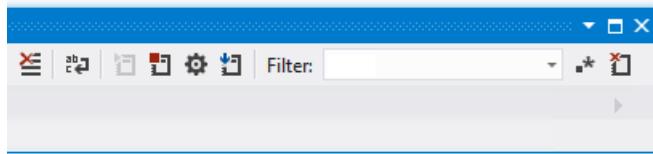
The screenshot shows the Microsoft Azure Logs tab in the Visual Studio Output window. The window title is "Output". The "Show output from" dropdown is set to "Windows Azure Logs - example1z". The log entries are as follows:

```
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
```

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

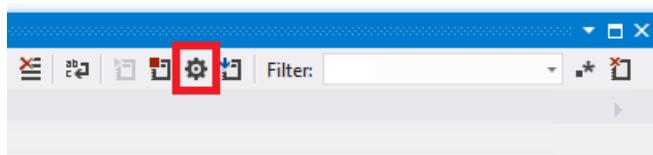
Logs Actions

- Open in Management Portal
- Stop Web Site
- Restart Web Site

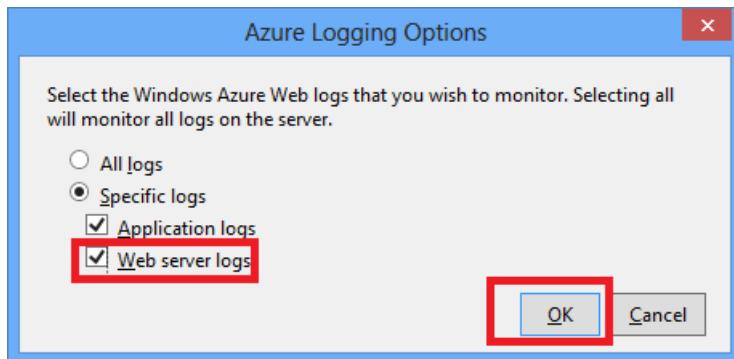
Web Site Settings

.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=2780cccd10d57b246 (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;
+WAIWebSiteSID=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;
+WAIWebSiteSID=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;
+WAIWebSiteSID=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

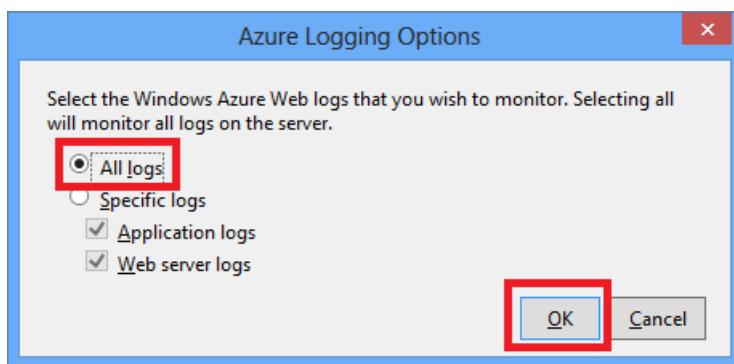
Logs Actions

- Open in Management Portal
- Stop Web Site
- Restart Web Site

Web Site Settings

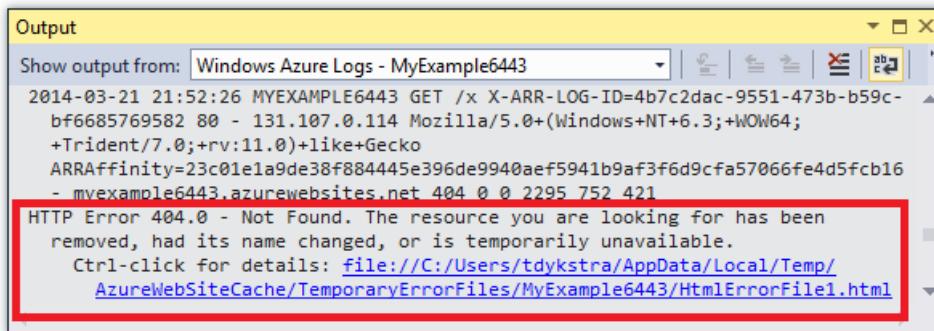
.NET Framework Version	v4.5
Web Server Logging	On
Detailed Error Messages	On
Failed Request Tracing	Off
Application Logging (File System)	Verbose

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.



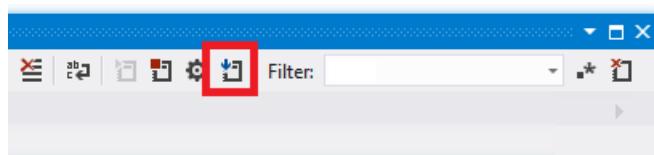
Control+click the link to see the log output formatted in a browser:

The screenshot shows an IIS Detailed Error page for a 404.0 Not Found error. The title bar says "IIS Detailed Error - 404....". The main content area has a red header "HTTP Error 404.0 - Not Found" and a message: "The resource you are looking for has been removed, had its name changed, or is temporarily unavailable." 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.", and "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.", and "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 about the request: Module ManagedPipelineHandler, Request http://example1z.azurewebsites.net:80/Home/Contactx, Notification ExecuteRequestHandler, Physical Path C:\DWASFiles\Sites\example1z\VVirtualDirectory0\site\wwwroot\Home\Contactx, Handler System.Web.Mvc.MvcHandler, Logon Method Anonymous, Error Code 0x00000000, and Logon User Anonymous. A "More Information:" section states that the error means the file or directory does not exist on the server and provides a link to "View more information >".

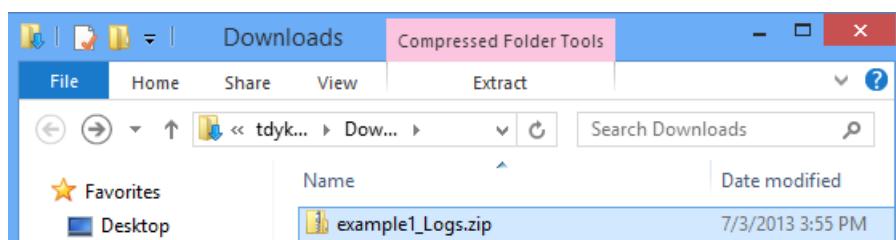
## Download file system logs

Any logs that you can monitor in the **Output** window can also be downloaded as a .zip file.

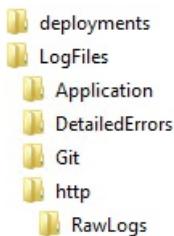
1. In the **Output** window, click **Download Streaming Logs**.



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**.

The screenshot shows the 'Configuration' tab of the Azure Web App window. At the top, there are 'Actions' buttons: 'Save' (highlighted with a red box) and 'Refresh'. Below that is a 'Web Site Settings' group. Under 'Web Site Settings', there are several dropdown menus:

- '.NET Framework Version': v4.5
- 'Web Server Logging': On
- 'Detailed Error Messages': On
- 'Failed Request Tracing': On (highlighted with a red box)
- 'Application Logging (File System)': Verbose

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.

**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

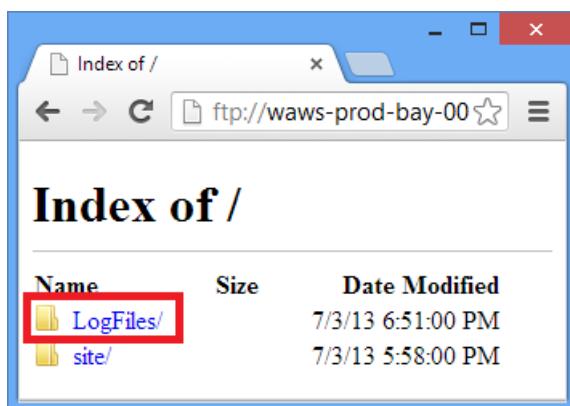
#### NOTE

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".

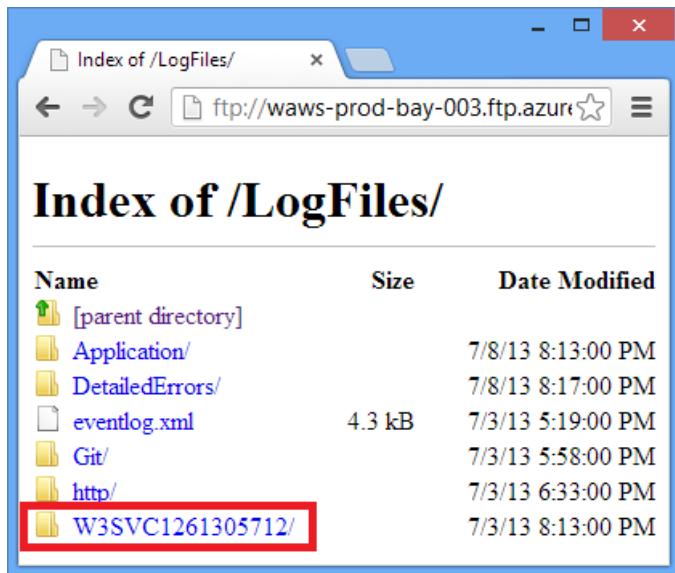
5. In a new browser window, go to the URL that is shown under **FTP hostname** or **FTPS hostname** in the **Overview** page for your app.
6. Sign in using the FTP credentials that you created earlier (including the app name prefix for the user name).

The browser shows the root folder of the app.

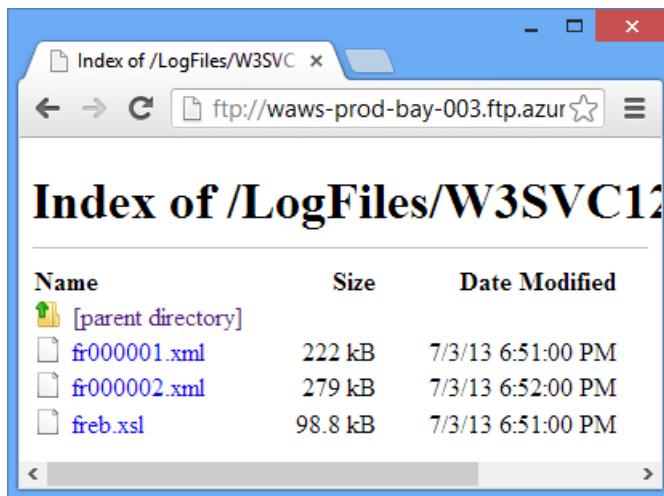
7. Open the *LogFiles* folder.



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

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 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 MSDN](#).
- [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

12/10/2019 • 12 minutes to read • [Edit Online](#)

In this article, you learn best practices and troubleshooting steps for [node 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 nodejs 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` \* 40 maxSockets/instance) 160 sockets per VM.

Example `agentKeepALive` configuration:

```
let keepaliveAgent = new Agent({
  maxSockets: 40,
  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 titled "... / wwwroot" showing four items: node\_modules, hostingstart.html, server.js, and web.config. Below the file browser is a command prompt window. The command prompt shows the following output:

```
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 server.js 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

▼ ▲

Use old console

```

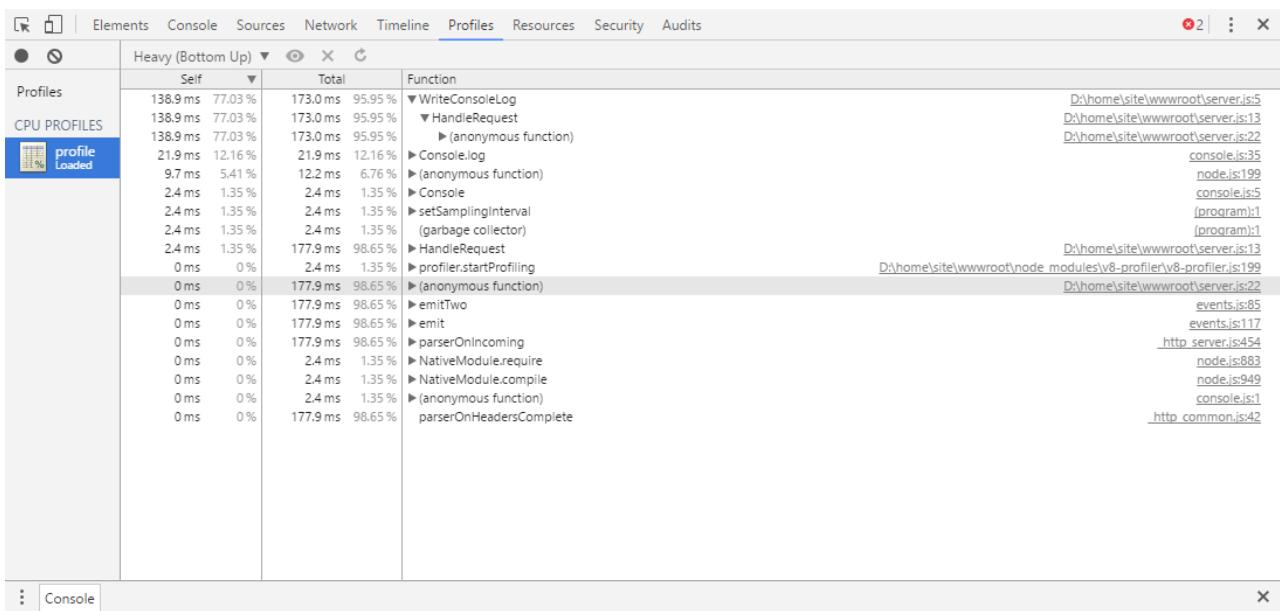
Kudu Remote Execution Console
Type 'exit' then hit 'enter' to get a new CMD process.
Type 'cls' to clear the console

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  $(maxNamedPipeConnectionRetry * namedPipeConnectionRetryDelay) / 1000$  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. Be sure you have a flat dependency structure and no duplicate dependencies by using npm3 to install your modules.
2. 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.
3. 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 maxConcurrentRequestsPerProcess 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

12/2/2019 • 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](#).

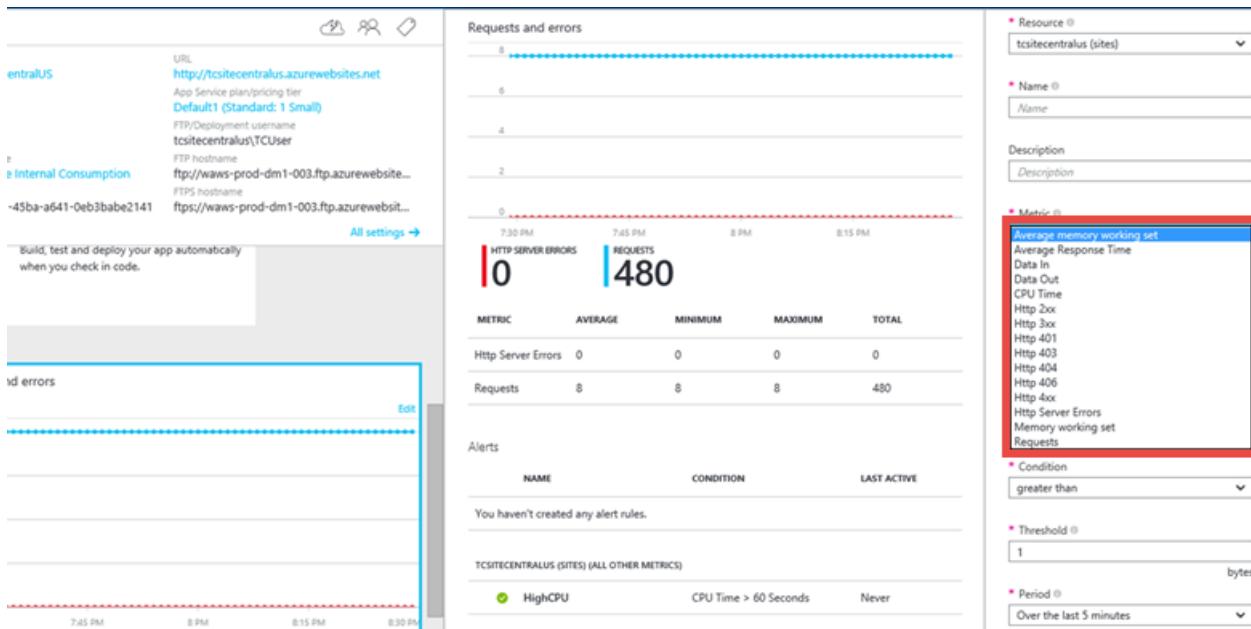
#### Monitor your app

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

12/2/2019 • 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

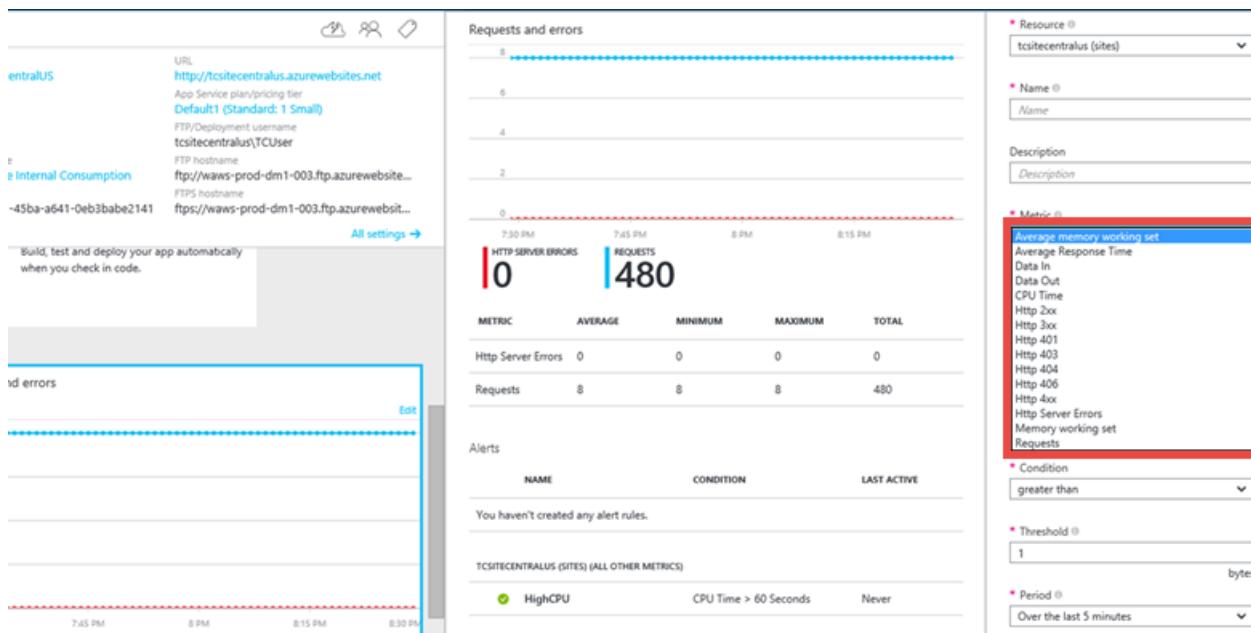
Microsoft Azure publicizes each time there is a service interruption or performance degradation. You can track the health of the service on the [Azure portal](#). For more information, see [Track service health](#).

#### 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](#).

# Application performance FAQs for Web Apps in Azure

1/3/2020 • 8 minutes to read • [Edit Online](#)

## NOTE

Some of the below guidelines might only work on Windows or Linux App Services. For example, Linux App Services run in 64-bit mode by default.

This article has answers to frequently asked questions (FAQs) about application performance issues for the [Web Apps feature of Azure App Service](#).

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

## Why is my app slow?

Multiple factors might contribute to slow app performance. For detailed troubleshooting steps, see [Troubleshoot slow web app performance](#).

## How do I troubleshoot a high CPU-consumption scenario?

In some high CPU-consumption scenarios, your app might truly require more computing resources. In that case, consider scaling to a higher service tier so the application gets all the resources it needs. Other times, high CPU consumption might be caused by a bad loop or by a coding practice. Getting insight into what's triggering increased CPU consumption is a two-part process. First, create a process dump, and then analyze the process dump. For more information, see [Capture and analyze a dump file for high CPU consumption for Web Apps](#).

## How do I troubleshoot a high memory-consumption scenario?

In some high memory-consumption scenarios, your app might truly require more computing resources. In that case, consider scaling to a higher service tier so the application gets all the resources it needs. Other times, a bug in the code might cause a memory leak. A coding practice also might increase memory consumption. Getting insight into what's triggering high memory consumption is a two-part process. First, create a process dump, and then analyze the process dump. Crash Diagnoser from the Azure Site Extension Gallery can efficiently perform both these steps. For more information, see [Capture and analyze a dump file for intermittent high memory for Web Apps](#).

## How do I automate App Service web apps by using PowerShell?

You can use PowerShell cmdlets to manage and maintain App Service web apps. In our blog post [Automate web apps hosted in Azure App Service by using PowerShell](#), we describe how to use Azure Resource Manager-based PowerShell cmdlets to automate common tasks. The blog post also has sample code for various web apps management tasks. For descriptions and syntax for all App Service web apps cmdlets, see [Az.Websites](#).

## How do I view my web app's event logs?

To view your web app's event logs:

1. Sign in to your [Kudu website](#).
2. In the menu, select **Debug Console > CMD**.
3. Select the **LogFiles** folder.
4. To view event logs, select the pencil icon next to **eventlog.xml**.
5. To download the logs, run the PowerShell cmdlet `Save-AzureWebSiteLog -Name webappname`.

## How do I capture a user-mode memory dump of my web app?

To capture a user-mode memory dump of your web app:

1. Sign in to your [Kudu website](#).
2. Select the **Process Explorer** menu.
3. Right-click the **w3wp.exe** process or your WebJob process.
4. Select **Download Memory Dump > Full Dump**.

## How do I view process-level info for my web app?

You have two options for viewing process-level information for your web app:

- In the Azure portal:
  1. Open the **Process Explorer** for the web app.
  2. To see the details, select the **w3wp.exe** process.
- In the Kudu console:
  1. Sign in to your [Kudu website](#).
  2. Select the **Process Explorer** menu.
  3. For the **w3wp.exe** process, select **Properties**.

## When I browse to my app, I see "Error 403 - This web app is stopped." How do I resolve this?

Three conditions can cause this error:

- The web app has reached a billing limit and your site has been disabled.
- The web app has been stopped in the portal.
- The web app has reached a resource quota limit that might apply to a Free or Shared scale service plan.

To see what is causing the error and to resolve the issue, follow the steps in [Web Apps: "Error 403 – This web app is stopped"](#).

## Where can I learn more about quotas and limits for various App Service plans?

For information about quotas and limits, see [App Service limits](#).

## How do I decrease the response time for the first request after idle time?

By default, web apps are unloaded if they are idle for a set period of time. This way, the system can conserve resources. The downside is that the response to the first request after the web app is unloaded is longer, to allow the web app to load and start serving responses. In Basic and Standard service plans, you can turn on the **Always On** setting to keep the app always loaded. This eliminates longer load times after the app is idle. To change the

## **Always On** setting:

1. In the Azure portal, go to your web app.
2. Select **Application settings**.
3. For **Always On**, select **On**.

## How do I turn on failed request tracing?

To turn on failed request tracing:

1. In the Azure portal, go to your web app.
2. Select **All Settings > Diagnostics Logs**.
3. For **Failed Request Tracing**, select **On**.
4. Select **Save**.
5. On the web app blade, select **Tools**.
6. Select **Visual Studio Online**.
7. If the setting is not **On**, select **On**.
8. Select **Go**.
9. Select **Web.config**.
10. In system.webServer, add this configuration (to capture a specific URL):

```
<system.webServer>
<tracing> <traceFailedRequests>
<remove path="*api*" />
<add path="*api*">
<traceAreas>
<add provider="ASP" verbosity="Verbose" />
<add provider="ASPNET" areas="Infrastructure,Module,Page,AppServices" verbosity="Verbose" />
<add provider="ISAPI Extension" verbosity="Verbose" />
<add provider="WWW Server" areas="Authentication,Security,Filter,StaticFile,CGI,Compression,
Cache,RequestNotifications,Module,FastCGI" verbosity="Verbose" />
</traceAreas>
<failureDefinitions statusCodes="200-999" />
</add> </traceFailedRequests>
</tracing>
```

11. To troubleshoot slow-performance issues, add this configuration (if the capturing request is taking more than 30 seconds):

```
<system.webServer>
<tracing> <traceFailedRequests>
<remove path="*" />
<add path="*">
<traceAreas> <add provider="ASP" verbosity="Verbose" />
<add provider="ASPNET" areas="Infrastructure,Module,Page,AppServices" verbosity="Verbose" />
<add provider="ISAPI Extension" verbosity="Verbose" />
<add provider="WWW Server" areas="Authentication,Security,Filter,StaticFile,CGI,Compression,
Cache,RequestNotifications,Module,FastCGI" verbosity="Verbose" />
</traceAreas>
<failureDefinitions timeTaken="00:00:30" statusCodes="200-999" />
</add> </traceFailedRequests>
</tracing>
```

12. To download the failed request traces, in the [portal](#), go to your website.
13. Select **Tools > Kudu > Go**.
14. In the menu, select **Debug Console > CMD**.
15. Select the **LogFiles** folder, and then select the folder with a name that starts with **W3SVC**.
16. To see the XML file, select the pencil icon.

## I see the message "Worker Process requested recycle due to 'Percent Memory' limit." How do I address this issue?

The maximum available amount of memory for a 32-bit process (even on a 64-bit operating system) is 2 GB. By default, the worker process is set to 32-bit in App Service (for compatibility with legacy web applications).

Consider switching to 64-bit processes so you can take advantage of the additional memory available in your Web Worker role. This triggers a web app restart, so schedule accordingly.

Also note that a 64-bit environment requires a Basic or Standard service plan. Free and Shared plans always run in a 32-bit environment.

For more information, see [Configure web apps in App Service](#).

## Why does my request time out after 230 seconds?

Azure Load Balancer has a default idle timeout setting of four minutes. This is generally a reasonable response time limit for a web request. If your web app requires background processing, we recommend using Azure WebJobs. The Azure web app can call WebJobs and be notified when background processing is finished. You can choose from multiple methods for using WebJobs, including queues and triggers.

WebJobs is designed for background processing. You can do as much background processing as you want in a WebJob. For more information about WebJobs, see [Run background tasks with WebJobs](#).

## ASP.NET Core applications that are hosted in App Service sometimes stop responding. How do I fix this issue?

A known issue with an earlier [Kestrel version](#) might cause an ASP.NET Core 1.0 app that's hosted in App Service to intermittently stop responding. You also might see this message: "The specified CGI Application encountered an error and the server terminated the process."

This issue is fixed in Kestrel version 1.0.2. This version is included in the ASP.NET Core 1.0.3 update. To resolve this issue, make sure you update your app dependencies to use Kestrel 1.0.2. Alternatively, you can use one of two workarounds that are described in the blog post [ASP.NET Core 1.0 slow perf issues in App Service web apps](#).

## I can't find my log files in the file structure of my web app. How can I find them?

If you use the Local Cache feature of App Service, the folder structure of the LogFiles and Data folders for your App Service instance are affected. When Local Cache is used, subfolders are created in the storage LogFiles and Data folders. The subfolders use the naming pattern "unique identifier" + time stamp. Each subfolder corresponds to a VM instance in which the web app is running or has run.

To determine whether you are using Local Cache, check your App Service **Application settings** tab. If Local Cache is being used, the app setting `WEBSITE_LOCAL_CACHE_OPTION` is set to `Always`.

If you are not using Local Cache and are experiencing this issue, submit a support request.

## I see the message "An attempt was made to access a socket in a way forbidden by its access permissions." How do I resolve this?

This error typically occurs if the outbound TCP connections on the VM instance are exhausted. In App Service, limits are enforced for the maximum number of outbound connections that can be made for each VM instance. For more information, see [Cross-VM numerical limits](#).

This error also might occur if you try to access a local address from your application. For more information, see [Local address requests](#).

For more information about outbound connections in your web app, see the blog post about [outgoing connections to Azure websites](#).

## How do I use Visual Studio to remote debug my App Service web app?

For a detailed walkthrough that shows you how to debug your web app by using Visual Studio, see [Remote debug your App Service web app](#).

# Deployment FAQs for Web Apps in Azure

12/2/2019 • 4 minutes to read • [Edit Online](#)

This article has answers to frequently asked questions (FAQs) about deployment issues for the [Web Apps feature of Azure App Service](#).

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

## I am just getting started with App Service web apps. How do I publish my code?

Here are some options for publishing your web app code:

- Deploy by using Visual Studio. If you have the Visual Studio solution, right-click the web application project, and then select **Publish**.
- Deploy by using an FTP client. In the Azure portal, download the publish profile for the web app that you want to deploy your code to. Then, upload the files to `\site\wwwroot` by using the same publish profile FTP credentials.

For more information, see [Deploy your app to App Service](#).

## I see an error message when I try to deploy from Visual Studio. How do I resolve this error?

If you see the following message, you might be using an older version of the SDK: "Error during deployment for resource 'YourResourceName' in resource group 'YourResourceGroup': MissingRegistrationForLocation: The subscription is not registered for the resource type 'components' in the location 'Central US'. Re-register for this provider in order to have access to this location."

To resolve this error, upgrade to the [latest SDK](#). If you see this message and you have the latest SDK, submit a support request.

## How do I deploy an ASP.NET application from Visual Studio to App Service?

The tutorial [Create your first ASP.NET web app in Azure in five minutes](#) shows you how to deploy an ASP.NET web application to a web app in App Service by using Visual Studio.

## What are the different types of deployment credentials?

App Service supports two types of credentials for local Git deployment and FTP/S deployment. For more information about how to configure deployment credentials, see [Configure deployment credentials for App Service](#).

## What is the file or directory structure of my App Service web app?

For information about the file structure of your App Service app, see [File structure in Azure](#).

## How do I resolve "FTP Error 550 - There is not enough space on the disk" when I try to FTP my files?

If you see this message, it's likely that you're running into a disk quota in the service plan for your web app. You might need to scale up to a higher service tier based on your disk space needs. For more information about pricing plans and resource limits, see [App Service pricing](#).

## How do I set up continuous deployment for my App Service web app?

You can set up continuous deployment from several resources, including Azure DevOps, OneDrive, GitHub, Bitbucket, Dropbox, and other Git repositories. These options are available in the portal. [Continuous deployment to App Service](#) is a helpful tutorial that explains how to set up continuous deployment.

## How do I troubleshoot issues with continuous deployment from GitHub and Bitbucket?

For help investigating issues with continuous deployment from GitHub or Bitbucket, see [Investigating continuous deployment](#).

## I can't FTP to my site and publish my code. How do I resolve this issue?

To resolve FTP issues:

1. Verify that you're entering the correct host name and credentials. For detailed information about different types of credentials and how to use them, see [Deployment credentials](#).
2. Verify that the FTP ports are not blocked by a firewall. The ports should have these settings:
  - FTP control connection port: 21
  - FTP data connection port: 989, 10001-10300

## How do I publish my code to App Service?

The Azure Quickstart is designed to help you deploy your app by using the deployment stack and method of your choice. To use the Quickstart, in the Azure portal, go to your app service, under **Deployment**, select **Quickstart**.

## Why does my app sometimes restart after deployment to App Service?

To learn about the circumstances under which an application deployment might result in a restart, see [Deployment vs. runtime issues](#). As the article describes, App Service deploys files to the wwwroot folder. It never directly restarts your app.

## How do I integrate Azure DevOps code with App Service?

You have two options for using continuous deployment with Azure DevOps:

- Use a Git project. Connect via App Service by using the Deployment Center.
- Use a Team Foundation Version Control (TFVC) project. Deploy by using the build agent for App Service.

Continuous code deployment for both these options depends on existing developer workflows and check-in procedures. For more information, see these articles:

- [Implement continuous deployment of your app to an Azure website](#)
- [Set up an Azure DevOps organization so it can deploy to a web app](#)

## How do I use FTP or FTPS to deploy my app to App Service?

For information about using FTP or FTPS to deploy your web app to App Service, see [Deploy your app to App Service by using FTP/S](#).

# Open-source technologies FAQs for Web Apps in Azure

12/2/2019 • 8 minutes to read • [Edit Online](#)

This article has answers to frequently asked questions (FAQs) about issues with open-source technologies for the [Web Apps feature of Azure App Service](#).

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

## How do I turn on PHP logging to troubleshoot PHP issues?

To turn on PHP logging:

1. Sign in to your [Kudu website](#).
2. In the top menu, select **Debug Console > CMD**.
3. Select the **Site** folder.
4. Select the **wwwroot** folder.
5. Select the + icon, and then select **New File**.
6. Set the file name to **.user.ini**.
7. Select the pencil icon next to **.user.ini**.
8. In the file, add this code: `log_errors=on`
9. Select **Save**.
10. Select the pencil icon next to **wp-config.php**.
11. Change the text to the following code:

```
//Enable WP_DEBUG modedefine('WP_DEBUG', true); //Enable debug logging to /wp-content/debug.logdefine('WP_DEBUG_LOG', true);  
//Suppress errors and warnings to screendefine('WP_DEBUG_DISPLAY', false); //Suppress PHP errors to screenini_set('display_errors', 0);
```

12. In the Azure portal, in the web app menu, restart your web app.

For more information, see [Enable WordPress error logs](#).

## How do I log Python application errors in apps that are hosted in App Service?

If Python encounters an error while starting your application, only a simple error page will be returned (e.g. "The page cannot be displayed because an internal server error has occurred").

To capture Python application errors:

1. In the Azure portal, in your web app, select **Settings**.
2. On the **Settings** tab, select **Application settings**.
3. Under **App settings**, enter the following key/value pair:
  - Key : WSGI\_LOG

- Value : D:\home\site\wwwroot\logs.txt (enter your choice of file name)

You should now see errors in the logs.txt file in the wwwroot folder.

## How do I change the version of the Node.js application that is hosted in App Service?

To change the version of the Node.js application, you can use one of the following options:

- In the Azure portal, use **App settings**.

1. In the Azure portal, go to your web app.
2. On the **Settings** blade, select **Application settings**.
3. In **App settings**, you can include WEBSITE\_NODE\_DEFAULT\_VERSION as the key, and the version of Node.js you want as the value.
4. Go to your [Kudu console](#).
5. To check the Node.js version, enter the following command:

```
node -v
```

- Modify the iisnode.yml file. Changing the Node.js version in the iisnode.yml file only sets the runtime environment that iisnode uses. Your Kudu cmd and others still use the Node.js version that is set in **App settings** in the Azure portal.

To set the iisnode.yml manually, create an iisnode.yml file in your app root folder. In the file, include the following line:

```
nodeProcessCommandLine: "D:\Program Files (x86)\nodejs\5.9.1\node.exe"
```

- Set the iisnode.yml file by using package.json during source control deployment. The Azure source control deployment process involves the following steps:

1. Moves content to the Azure web app.
2. Creates a default deployment script, if there isn't one (deploy.cmd, .deployment files) in the web app root folder.
3. Runs a deployment script in which it creates an iisnode.yml file if you mention the Node.js version in the package.json file > engine `"engines": {"node": "5.9.1", "npm": "3.7.3"}`
4. The iisnode.yml file has the following line of code:

```
nodeProcessCommandLine: "D:\Program Files (x86)\nodejs\5.9.1\node.exe"
```

I see the message "Error establishing a database connection" in my WordPress app that's hosted in App Service. How do I troubleshoot this?

If you see this error in your Azure WordPress app, to enable php\_errors.log and debug.log, complete the steps detailed in [Enable WordPress error logs](#).

When the logs are enabled, reproduce the error, and then check the logs to see if you are running out of connections:

```
[09-Oct-2015 00:03:13 UTC] PHP Warning: mysqli_real_connect(): (HY000/1226): User 'abcdefghijklm' has exceeded the 'max_user_connections' resource (current value: 4) in D:\home\site\wwwroot\wp-includes\wp-db.php on line 1454
```

If you see this error in your debug.log or php\_errors.log files, your app is exceeding the number of connections. If you're hosting on ClearDB, verify the number of connections that are available in your [service plan](#).

## How do I debug a Node.js app that's hosted in App Service?

1. Go to your [Kudu console](#).
2. Go to your application logs folder (D:\home\LogFiles\Application).
3. In the logging\_errors.txt file, check for content.

## How do I install native Python modules in an App Service web app or API app?

Some packages might not install by using pip in Azure. The package might not be available on the Python Package Index, or a compiler might be required (a compiler is not available on the computer that is running the web app in App Service). For information about installing native modules in App Service web apps and API apps, see [Install Python modules in App Service](#).

## How do I deploy a Django app to App Service by using Git and the new version of Python?

For information about installing Django, see [Deploying a Django app to App Service](#).

## Where are the Tomcat log files located?

For Azure Marketplace and custom deployments:

- Folder location: D:\home\site\wwwroot\bin\apache-tomcat-8.0.33\logs
- Files of interest:
  - catalina.yyyy-mm-dd.log
  - host-manager.yyyy-mm-dd.log
  - localhost.yyyy-mm-dd.log
  - manager.yyyy-mm-dd.log
  - site\_access\_log.yyyy-mm-dd.log

For portal **App settings** deployments:

- Folder location: D:\home\LogFiles
- Files of interest:
  - catalina.yyyy-mm-dd.log
  - host-manager.yyyy-mm-dd.log
  - localhost.yyyy-mm-dd.log
  - manager.yyyy-mm-dd.log
  - site\_access\_log.yyyy-mm-dd.log

## How do I troubleshoot JDBC driver connection errors?

You might see the following message in your Tomcat logs:

```
The web application[ROOT] registered the JDBC driver [com.mysql.jdbc.Driver] but failed to unregister it when the web application was stopped. To prevent a memory leak, the JDBC Driver has been forcibly unregistered
```

To resolve the error:

1. Remove the sqljdbc\*.jar file from your app/lib folder.
2. If you are using the custom Tomcat or Azure Marketplace Tomcat web server, copy this jar file to the Tomcat lib folder.
3. If you are enabling Java from the Azure portal (select **Java 1.8 > Tomcat server**), copy the sqljdbc.\* jar file in the folder that's parallel to your app. Then, add the following classpath setting to the web.config file:

```
<httpPlatform>
<environmentVariables>
<environmentVariableName ="JAVA_OPTS" value=" -Djava.net.preferIPv4Stack=true
-Xms128M -classpath %CLASSPATH%;[Path to the sqljdbc*.jarfile]" />
</environmentVariables>
</httpPlatform>
```

## Why do I see errors when I attempt to copy live log files?

If you try to copy live log files for a Java app (for example, Tomcat), you might see this FTP error:

```
Error transferring file [filename] Copying files from remote side failed.

The process cannot access the file because it is being used by another process.
```

The error message might vary, depending on the FTP client.

All Java apps have this locking issue. Only Kudu supports downloading this file while the app is running.

Stopping the app allows FTP access to these files.

Another workaround is to write a WebJob that runs on a schedule and copies these files to a different directory. For a sample project, see the [CopyLogsJob](#) project.

## Where do I find the log files for Jetty?

For Marketplace and custom deployments, the log file is in the D:\home\site\wwwroot\bin\jetty-distribution-9.1.2.v20140210\logs folder. Note that the folder location depends on the version of Jetty you are using. For example, the path provided here is for Jetty 9.1.2. Look for jetty\_YYYY\_MM\_DD.stderrout.log.

For portal App Setting deployments, the log file is in D:\home\LogFiles. Look for jetty\_YYYY\_MM\_DD.stderrout.log

## Can I send email from my Azure web app?

App Service doesn't have a built-in email feature. For some good alternatives for sending email from your app, see this [Stack Overflow discussion](#).

## Why does my WordPress site redirect to another URL?

If you have recently migrated to Azure, WordPress might redirect to the old domain URL. This is caused by a setting in the MySQL database.

WordPress Buddy+ is an Azure Site Extension that you can use to update the redirection URL directly in the database. For more information about using WordPress Buddy+, see [WordPress tools and MySQL migration with WordPress Buddy+](#).

Alternatively, if you prefer to manually update the redirection URL by using SQL queries or PHPMyAdmin, see [WordPress: Redirecting to wrong URL](#).

## How do I change my WordPress sign-in password?

If you have forgotten your WordPress sign-in password, you can use WordPress Buddy+ to update it. To reset your password, install the WordPress Buddy+ Azure Site Extension, and then complete the steps described in [WordPress tools and MySQL migration with WordPress Buddy+](#).

## I can't sign in to WordPress. How do I resolve this?

If you find yourself locked out of WordPress after recently installing a plugin, you might have a faulty plugin. WordPress Buddy+ is an Azure Site Extension that can help you disable plugins in WordPress. For more information, see [WordPress tools and MySQL migration with WordPress Buddy+](#).

## How do I migrate my WordPress database?

You have multiple options for migrating the MySQL database that's connected to your WordPress website:

- Developers: Use the [command prompt](#) or [PHPMyAdmin](#)
- Non-developers: Use [WordPress Buddy+](#)

## How do I help make WordPress more secure?

To learn about security best practices for WordPress, see [Best practices for WordPress security in Azure](#).

## I am trying to use PHPMyAdmin, and I see the message "Access denied." How do I resolve this?

You might experience this issue if the MySQL in-app feature isn't running yet in this App Service instance. To resolve the issue, try to access your website. This starts the required processes, including the MySQL in-app process. To verify that MySQL in-app is running, in Process Explorer, ensure that mysqld.exe is listed in the processes.

After you ensure that MySQL in-app is running, try to use PHPMyAdmin.

## I get an HTTP 403 error when I try to import or export my MySQL in-app database by using PHPMyadmin. How do I resolve this?

If you are using an older version of Chrome, you might be experiencing a known bug. To resolve the issue, upgrade to a newer version of Chrome. Also try using a different browser, like Internet Explorer or Microsoft Edge, where the issue does not occur.

# Configuration and management FAQs for Web Apps in Azure

2/26/2020 • 15 minutes to read • [Edit Online](#)

This article has answers to frequently asked questions (FAQs) about configuration and management issues for the [Web Apps feature of Azure App Service](#).

If your Azure issue is not addressed in this article, visit the Azure forums on [MSDN and Stack Overflow](#). You can post your issue in these forums, or post to [@AzureSupport on Twitter](#). You also can submit an Azure support request. To submit a support request, on the [Azure support](#) page, select **Get support**.

## Are there limitations I should be aware of if I want to move App Service resources?

If you plan to move App Service resources to a new resource group or subscription, there are a few limitations to be aware of. For more information, see [App Service limitations](#).

## How do I use a custom domain name for my web app?

For answers to common questions about using a custom domain name with your Azure web app, see our seven-minute video [Add a custom domain name](#). The video offers a walkthrough of how to add a custom domain name. It describes how to use your own URL instead of the \*.azurewebsites.net URL with your App Service web app. You also can see a detailed walkthrough of [how to map a custom domain name](#).

## How do I purchase a new custom domain for my web app?

To learn how to purchase and set up a custom domain for your App Service web app, see [Buy and configure a custom domain name in App Service](#).

## How do I upload and configure an existing SSL certificate for my web app?

To learn how to upload and set up an existing custom SSL certificate, see [Add an SSL certificate to your App Service app](#).

## How do I purchase and configure a new SSL certificate in Azure for my web app?

To learn how to purchase and set up an SSL certificate for your App Service web app, see [Add an SSL certificate to your App Service app](#).

## How do I move Application Insights resources?

Currently, Azure Application Insights doesn't support the move operation. If your original resource group includes an Application Insights resource, you cannot move that resource. If you include the Application Insights resource when you try to move an App Service app, the entire move operation fails. However, Application Insights and the App Service plan do not need to be in the same resource group as the app for the app to function correctly.

For more information, see [App Service limitations](#).

## Where can I find a guidance checklist and learn more about resource move operations?

[App Service limitations](#) shows you how to move resources to either a new subscription or to a new resource group in the same subscription. You can get information about the resource move checklist, learn which services support the move operation, and learn more about App Service limitations and other topics.

## How do I set the server time zone for my web app?

To set the server time zone for your web app:

1. In the Azure portal, in your App Service subscription, go to the **Application settings** menu.
2. Under **App settings**, add this setting:
  - Key = WEBSITE\_TIME\_ZONE
  - Value = *The time zone you want*
3. Select **Save**.

For the App services that run on Windows, see the **Timezone** column in the [Default Time Zones](#) article for accepted values. For the App services that run on Linux, set the **TZ database name** as the time zone value. Here is an example of TZ database name: America/Adak.

## Why do my continuous WebJobs sometimes fail?

By default, web apps are unloaded if they are idle for a set period of time. This lets the system conserve resources. In Basic and Standard plans, you can turn on the **Always On** setting to keep the web app loaded all the time. If your web app runs continuous WebJobs, you should turn on **Always On**, or the WebJobs might not run reliably. For more information, see [Create a continuously running WebJob](#).

## How do I get the outbound IP address for my web app?

To get the list of outbound IP addresses for your web app:

1. In the Azure portal, on your web app blade, go to the **Properties** menu.
2. Search for **outbound ip addresses**.

The list of outbound IP addresses appears.

To learn how to get the outbound IP address if your website is hosted in an App Service Environment, see [Outbound network addresses](#).

## How do I get a reserved or dedicated inbound IP address for my web app?

To set up a dedicated or reserved IP address for inbound calls made to your Azure app website, install and configure an IP-based SSL certificate.

Note that to use a dedicated or reserved IP address for inbound calls, your App Service plan must be in a Basic or higher service plan.

## Can I export my App Service certificate to use outside Azure, such as for a website hosted elsewhere?

Yes, you can export them to use outside Azure. For more information, see [FAQs for App Service certificates and custom domains](#).

## Can I export my App Service certificate to use with other Azure cloud services?

The portal provides a first-class experience for deploying an App Service certificate through Azure Key Vault to App Service apps. However, we have been receiving requests from customers to use these certificates outside the App Service platform, for example, with Azure Virtual Machines. To learn how to create a local PFX copy of your App Service certificate so you can use the certificate with other Azure resources, see [Create a local PFX copy of an App Service certificate](#).

For more information, see [FAQs for App Service certificates and custom domains](#).

## Why do I see the message "Partially Succeeded" when I try to back up my web app?

A common cause of backup failure is that some files are in use by the application. Files that are in use are locked while you perform the backup. This prevents these files from being backed up and might result in a "Partially Succeeded" status. You can potentially prevent this from occurring by excluding files from the backup process. You can choose to back up only what is needed. For more information, see [Backup just the important parts of your site with Azure web apps](#).

## How do I remove a header from the HTTP response?

To remove the headers from the HTTP response, update your site's web.config file. For more information, see [Remove standard server headers on your Azure websites](#).

## Is App Service compliant with PCI Standard 3.0 and 3.1?

Currently, the Web Apps feature of Azure App Service is in compliance with PCI Data Security Standard (DSS) version 3.0 Level 1. PCI DSS version 3.1 is on our roadmap. Planning is already underway for how adoption of the latest standard will proceed.

PCI DSS version 3.1 certification requires disabling Transport Layer Security (TLS) 1.0. Currently, disabling TLS 1.0 is not an option for most App Service plans. However, If you use App Service Environment or are willing to migrate your workload to App Service Environment, you can get greater control of your environment. This involves disabling TLS 1.0 by contacting Azure Support. In the near future, we plan to make these settings accessible to users.

For more information, see [Microsoft Azure App Service web app compliance with PCI Standard 3.0 and 3.1](#).

## How do I use the staging environment and deployment slots?

In Standard and Premium App Service plans, when you deploy your web app to App Service, you can deploy to a separate deployment slot instead of to the default production slot. Deployment slots are live web apps that have their own host names. Web app content and configuration elements can be swapped between two deployment slots, including the production slot.

For more information about using deployment slots, see [Set up a staging environment in App Service](#).

## How do I access and review WebJob logs?

To review WebJob logs:

1. Sign in to your [Kudu website](#).
2. Select the WebJob.

3. Select the **Toggle Output** button.
4. To download the output file, select the **Download** link.
5. For individual runs, select **Individual Invoke**.
6. Select the **Toggle Output** button.
7. Select the download link.

I'm trying to use Hybrid Connections with SQL Server. Why do I see the message "System.OverflowException: Arithmetic operation resulted in an overflow"?

If you use Hybrid Connections to access SQL Server, a Microsoft .NET update on May 10, 2016, might cause connections to fail. You might see this message:

```
Exception: System.Data.Entity.Core.EntityException: The underlying provider failed on Open. ->
System.OverflowException: Arithmetic operation resulted in an overflow. or (64 bit Web app)
System.OverflowException: Array dimensions exceeded supported range, at
System.Data.SqlClient.TdsParser.ConsumePreLoginHandshake
```

## Resolution

The exception was caused by an issue with the Hybrid Connection Manager that has since been fixed. Be sure to [update your Hybrid Connection Manager](#) to resolve this issue.

## How do I add a URL rewrite rule?

To add a URL rewrite rule, create a web.config file with the relevant config entries in the **wwwroot** folder. For more information, see [Azure App Services: Understanding URL rewrite](#).

## How do I control inbound traffic to App Service?

At the site level, you have two options for controlling inbound traffic to App Service:

- Turn on dynamic IP restrictions. To learn how to turn on dynamic IP restrictions, see [IP and domain restrictions for Azure websites](#).
- Turn on Module Security. To learn how to turn on Module Security, see [ModSecurity web application firewall on Azure websites](#).

If you use App Service Environment, you can use [Barracuda firewall](#).

## How do I block ports in an App Service web app?

In the App Service shared tenant environment, it is not possible to block specific ports because of the nature of the infrastructure. TCP ports 4020, 4022, and 4024 also might be open for Visual Studio remote debugging.

In App Service Environment, you have full control over inbound and outbound traffic. You can use Network Security Groups to restrict or block specific ports. For more information about App Service Environment, see [Introducing App Service Environment](#).

## How do I capture an F12 trace?

You have two options for capturing an F12 trace:

- F12 HTTP trace
- F12 console output

## F12 HTTP trace

1. In Internet Explorer, go to your website. It's important to sign in before you do the next steps. Otherwise, the F12 trace captures sensitive sign-in data.
2. Press F12.
3. Verify that the **Network** tab is selected, and then select the green **Play** button.
4. Do the steps that reproduce the issue.
5. Select the red **Stop** button.
6. Select the **Save** button (disk icon), and save the HAR file (in Internet Explorer and Microsoft Edge) or right-click the HAR file, and then select **Save as HAR with content** (in Chrome).

## F12 console output

1. Select the **Console** tab.
2. For each tab that contains more than zero items, select the tab (**Error**, **Warning**, or **Information**). If the tab isn't selected, the tab icon is gray or black when you move the cursor away from it.
3. Right-click in the message area of the pane, and then select **Copy all**.
4. Paste the copied text in a file, and then save the file.

To view an HAR file, you can use the [HAR viewer](#).

## Why do I get an error when I try to connect an App Service web app to a virtual network that is connected to ExpressRoute?

If you try to connect an Azure web app to a virtual network that's connected to Azure ExpressRoute, it fails. The following message appears: "Gateway is not a VPN gateway."

Currently, you cannot have point-to-site VPN connections to a virtual network that is connected to ExpressRoute. A point-to-site VPN and ExpressRoute cannot coexist for the same virtual network. For more information, see [ExpressRoute and site-to-site VPN connections limits and limitations](#).

## How do I connect an App Service web app to a virtual network that has a static routing (policy-based) gateway?

Currently, connecting an App Service web app to a virtual network that has a static routing (policy-based) gateway is not supported. If your target virtual network already exists, it must have point-to-site VPN enabled, with a dynamic routing gateway, before it can be connected to an app. If your gateway is set to static routing, you cannot enable a point-to-site VPN.

For more information, see [Integrate an app with an Azure virtual network](#).

## In my App Service Environment, why can I create only one App Service plan, even though I have two workers available?

To provide fault tolerance, App Service Environment requires that each worker pool needs at least one additional compute resource. The additional compute resource cannot be assigned a workload.

For more information, see [How to create an App Service Environment](#).

## Why do I see timeouts when I try to create an App Service Environment?

Sometimes, creating an App Service Environment fails. In that case, you see the following error in the Activity logs:

```
ResourceID: /subscriptions/{SubscriptionID}/resourceGroups/Default-Networking/providers/Microsoft.Web/hostingEnvironments/{ASEname}
Error:{ "error": { "code": "ResourceDeploymentFailure", "message": "The resource provision operation did not complete within the allowed timeout period." }}
```

To resolve this, make sure that none of the following conditions are true:

- The subnet is too small.
- The subnet is not empty.
- ExpressRoute prevents the network connectivity requirements of an App Service Environment.
- A bad Network Security Group prevents the network connectivity requirements of an App Service Environment.
- Forced tunneling is turned on.

For more information, see [Frequent issues when deploying \(creating\) a new Azure App Service Environment](#).

## Why can't I delete my App Service plan?

You can't delete an App Service plan if any App Service apps are associated with the App Service plan. Before you delete an App Service plan, remove all associated App Service apps from the App Service plan.

## How do I schedule a WebJob?

You can create a scheduled WebJob by using Cron expressions:

1. Create a settings.job file.
2. In this JSON file, include a schedule property by using a Cron expression:

```
{ "schedule": "{second}  
{minute} {hour} {day}  
{month} {day of the week}" }
```

For more information about scheduled WebJobs, see [Create a scheduled WebJob by using a Cron expression](#).

## How do I perform penetration testing for my App Service app?

To perform penetration testing, [submit a request](#).

## How do I configure a custom domain name for an App Service web app that uses Traffic Manager?

To learn how to use a custom domain name with an App Service app that uses Azure Traffic Manager for load balancing, see [Configure a custom domain name for an Azure web app with Traffic Manager](#).

## My App Service certificate is flagged for fraud. How do I resolve this?

### NOTE

This article has been updated to use the new Azure PowerShell Az module. You can still use the AzureRM module, which will continue to receive bug fixes until at least December 2020. To learn more about the new Az module and AzureRM compatibility, see [Introducing the new Azure PowerShell Az module](#). For Az module installation instructions, see [Install Azure PowerShell](#).

During the domain verification of an App Service certificate purchase, you might see the following 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, please contact Azure Support."

As the message indicates, this fraud verification process might take up to 24 hours to complete. During this time, you'll continue to see the message.

If your App Service certificate continues to show this message after 24 hours, please run the following PowerShell script. The script contacts the [certificate provider](#) directly to resolve the issue.

```
Connect-AzAccount
Set-AzContext -SubscriptionId <subId>
$actionProperties = @{
    "Name"= "<Customer Email Address>"
};
Invoke-AzResourceAction -ResourceGroupName "<App Service Certificate Resource Group Name>" -ResourceType Microsoft.CertificateRegistration/certificateOrders -ResourceName "<App Service Certificate Resource Name>" -Action resendRequestEmails -Parameters $actionProperties -ApiVersion 2015-08-01 -Force
```

## How do authentication and authorization work in App Service?

For detailed documentation for authentication and authorization in App Service, see docs for various identify provider sign-ins:

- [Azure Active Directory](#)
- [Facebook](#)
- [Google](#)
- [Microsoft Account](#)
- [Twitter](#)

## How do I redirect the default \*.azurewebsites.net domain to my Azure web app's custom domain?

When you create a new website by using Web Apps in Azure, a default *sitename.azurewebsites.net* domain is assigned to your site. If you add a custom host name to your site and don't want users to be able to access your default \*.azurewebsites.net domain, you can redirect the default URL. To learn how to redirect all traffic from your website's default domain to your custom domain, see [Redirect the default domain to your custom domain in Azure web apps](#).

## How do I determine which version of .NET version is installed in App Service?

The quickest way to find the version of Microsoft .NET that's installed in App Service is by using the Kudu console. You can access the Kudu console from the portal or by using the URL of your App Service app. For detailed instructions, see [Determine the installed .NET version in App Service](#).

## Why isn't Autoscale working as expected?

If Azure Autoscale hasn't scaled in or scaled out the web app instance as you expected, you might be running into a scenario in which we intentionally choose not to scale to avoid an infinite loop due to "flapping." This usually happens when there isn't an adequate margin between the scale-out and scale-in thresholds. To learn how to avoid "flapping" and to read about other Autoscale best practices, see [Autoscale best practices](#).

## Why does Autoscale sometimes scale only partially?

Autoscale is triggered when metrics exceed preconfigured boundaries. Sometimes, you might notice that the capacity is only partially filled compared to what you expected. This might occur when the number of instances you want are not available. In that scenario, Autoscale partially fills in with the available number of instances. Autoscale then runs the rebalance logic to get more capacity. It allocates the remaining instances. Note that this might take a few minutes.

If you don't see the expected number of instances after a few minutes, it might be because the partial refill was enough to bring the metrics within the boundaries. Or, Autoscale might have scaled down because it reached the lower metrics boundary.

If none of these conditions apply and the problem persists, submit a support request.

## How do I turn on HTTP compression for my content?

To turn on compression both for static and dynamic content types, add the following code to the application-level web.config file:

```
<system.webServer>
  <urlCompression doStaticCompression="true" doDynamicCompression="true" />
</system.webServer>
```

You also can specify the specific dynamic and static MIME types that you want to compress. For more information, see our response to a forum question in [httpCompression settings on a simple Azure website](#).

## How do I migrate from an on-premises environment to App Service?

To migrate sites from Windows and Linux web servers to App Service, you can use Azure App Service Migration Assistant. The migration tool creates web apps and databases in Azure as needed, and then publishes the content. For more information, see [Azure App Service Migration Assistant](#).